QoE-Centric Network-Assisted Delivery of Adaptive Video Streaming Services

QoE-centrische en netwerkgedreven aflevering van adaptieve videodiensten

Stefano Petrangeli

Promotoren: prof. dr. ir. F. De Turck, dr. ir. T. Wauters
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. B. Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017 - 2018

ISBN 978-94-6355-097-0
NUR 986, 988
Wettelijk depot: D/2018/10.500/15

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Leden van de examencommissie:

prof. dr. ir. Filip De Turck (promotor)
Universiteit Gent – imec
dr. ir. Tim Wauters (promotor)
Universiteit Gent – imec

prof. dr. ir. Daniël De Zutter (voorzitter)
Universiteit Gent – imec
dr. ir. Glenn Van Wallendael
Universiteit Gent – imec
prof. dr. Jeroen Famaey
Universiteit Antwerpen – imec
prof. dr. ir. Danny De Vleeschauwer
Nokia Bell Labs – Universiteit Gent
prof. dr. Christian Timmerer
Alpen-Adria-Universität Klagenfurt – Bitmovin
dr. ir. Jürgen Slowack
Barco

Proefschrift tot het behalen van de graad van
Doctor in de ingenieurswetenschappen:
computerwetenschappen
Academiejaar 2017-2018

Onder de maan schuift de lange rivier
Over de lange rivier schuift moede de maan
Onder de maan op de lange rivier schuift de kano naar zee

Langs het hoogriet
langs de laagwei
schuift de kano naar zee
schuift met de schuivende maan de kano naar zee
Zo zijn ze gezellen naar zee de kano de maan en de man
Waarom schuiven de maan en de man getween gedwee naar de zee

Acknowledgments

Five incredible years are enclosed in the pages of this book. Five years made of incredible moments, a lot of work, discoveries, travels, major setbacks, great encounters: probably the best of my adult life so far. All of this would have never been possible without the many good people that supported and helped me during my PhD in the beautiful city of Gent.

First and foremost, I would like to thank my supervisors Filip and Tim. Filip, I cannot fully express my gratitude for the opportunity you gave me. This experience has meant all my entire life in the last five years; I hope I was able to give back at least a fraction of what I received. Thanks for the guidance and support during my PhD, not just from the technical but especially from the personal point of view. Being far from home is a challenge that I was able to overcome also because of your presence. Tim, thanks for the continuous guidance, the ideas, for steering my PhD till the very end. Without your help and example, I would have never been able to complete this thesis.

A big thanks goes also to Jeroen and Steven, who helped me a lot at the beginning of my PhD. As the saying goes, well begun is half done, and it proved to be true in my case. Thanks to Jeroen "the-kangaroo-whisperer" for being such a good colleague and my favorite co-author, and to Raf for the nice research experience during the V-FORCE project.

A great PhD would be impossible without great colleagues, and I can boast I had many: Bram, Jerico, Jeroen S, Leandro, Maxim, Thomas, Merlijn, Niels, Olivier, Philip, Piet, Sander, Steven B, Steven VC, Thijs, Tim, Wim, Dries, Maria, Andrés, Roberto, Rafael, Andy, Laurens, Enri, Lucas: thanks to all of you! An honorable mention to Maxim, Niels and Thomas for having taught me a lot about Flemish habits, expressions and singing traditions and for the legendary NOMS 2014 together. A great thanks to Martine and Davinia, whose constant work, efforts and kindness allowed my PhD to smoothly proceed since the real beginning. I would like to express my heartfelt thanks to Merlijn "Merlino" for the interesting lunch discussions. Here is my top three: (i) debate around the usage of glass versus ceramic cups for drinking wine, beer, milk and coffee; (ii) the weird Japanese show that anticipated (and outclassed in so many ways...) the big brother and (iii) the quantum time machine "everybody can build in a garage". I hope Facebook ads will soon provide you with the love you so much deserve...

Thanks to Vishy for the great learning experience that was my internship and all the team in Adobe for the good time I had there. Mara, Sharan, Lakhan, thanks for all the fun we had together and for making my stay in the US memorable, despite the difficult moments. My internship would have been completely different

without you guys.

My deepest, sincere gratitude goes to an incredible group of friends and companions that has been my Family during the last five years: Catalina, Mario, Khaled, Mel, Leandro, Rafa, Tim, Andrés, Tiago and Silvia. Discovering this part of our lives together has been an experience that has transcended a normal friendship. You are the reason why Gent feels truly like Home. Khaled and Tiago, I owe you everything I was able to accomplish during my stay in Gent. Meeting you has been such a life turning point that I cannot possibly explain with simple words. It meant everything for me.

Un grande grazie a tutti i Muppets: Ray Charles, er perla, Valerio B(estia), er faina, Lorenzo, Pascal, er bussola, Gabriele, Daniele. Conosco molti di voi da quasi vent'anni, ed alcuni da quasi trenta. Inutile quindi nascondere il bene che vi voglio! Siamo stati ciascuno la salvezza (e la condanna J) dell'altro, e vi ringrazio di avermi sempre fatto essere me stesso. Meritano una menzione speciale Fabio e Valerio, che mi sono sempre stati accanto nei tantissimi bei momenti ed in quelli più rari e difficili. Sapere di poter contare su degli amici come voi ha permesso a questa tesi di essere scritta. Al mio amico e compagno di banco Beppe, la cui amicizia pluridecennale è una parte integrante del mio carattere. Nonostante tu non sappia la differenza tra neoplatonici ed aristotelici, almeno conosci bene le regole di ingaggio in prossimità di un mercato...

A tutti i miei zii, a tutte le mie cugine e cugini, a Vittorio: i pranzi di Natale, il chiasso, le tombolate infinite, le patate d'Avezzano, il finto cane regalato a Nonna, i brindisi a Berlusconi sono momenti indimenticabili. Nonostante la distanza, con voi mi sento sempre a Casa. A Zio Sandokan che mi ha fatto diventare Romanista e a Zia Patrizia che si è sempre impegnata a mostrarmi il lato nascosto della luna.

Alle mie nipotine Gaia, Viviana e Ludovica (e Flavio!), la cui spensieratezza ed i cui sorrisi sono il simbolo che, nonostante tutto e tutti, il mondo va avanti.

"Alla compagna di viaggio, i tuoi occhi il più bel paesaggio, fan sembrare più corto il cammino". A Giovanna Diletta che ha condiviso con me questo lungo viaggio durato cinque anni. Di momenti difficili ne abbiamo passati tanti, eppure rifarei tutto quello che ho fatto. Perché, nonostante tutto, siamo qui, assieme. Amo la tua intelligenza e lo splendore dei tuoi occhi, e non vedo l'ora di proseguire questa dolce passeggiata assieme.

Ai miei Nonni, che mi hanno insegnato che le cose più importanti non si imparano sui libri. Siete stati l'esempio più puro che abbia mai avuto nella mia vita. A Nonno Mario, che mi ha instillato l'amore per la scienza, la curiosità, la matematica che non potevo capire, Galileo, i satelliti di Giove e tanto altro.

Infine, a mia Sorella, a mio Padre, a mia Madre. Mi avete spinto dove non potevo arrivare, siete stati il coraggio che non credevo di possedere, il sorriso nei giorni difficili, l'appoggio prima della caduta. In altre parole, tutta la mia Famiglia. Vi appartengo come l'inchiostro a queste pagine.

Gent, March 2018
Stefano

Table of Contents

List of Figures

List of Tables

List of Acronyms

0-9

3GPP 3rd Generation Partnership Project

C

CCN Content Centric Networking
CDN Content Delivery Network

D

DANE DASH-Aware Network Element
DSCP Differentiated Service Code Point

E

EPC Evolved Packet Core

F

FINEAS Fair In-Network Enhanced Adaptive Streaming

G

GCC Google Congestion Control

H

HAS HTTP Adaptive Streaming

I

ICN Information Centric Networking
IXP Internet Exchange Point
ILP Integer Linear Programming
ISP Internet Service Provider
IoT Internet of Things

M

MCU Multipoint Conferencing Unit
MDP Markov Decision Process
MPEG-DASH MPEG Dynamic Adaptive Streaming over HTTP
MSS Microsoft ISS Smooth Streaming
ML Machine Learning
MOS Mean Opinion Score
MPTCP Multi-Path TCP
MPD Media Presentation Description
MPEG-SAND MPEG Server and Network-Assisted DASH

O

OTT Over-the-Top

P

PD Progressive Download
PED Parameter Enhancing Delivery

PER Parameter Enhancing Reception
PSNR Peak-Signal-to-Noise-Ratio

Q

QoE Quality of Experience
QoS Quality of Service
QUIC Quick UDP Internet Connection

R

REMB Receiver Estimated Maximum Bitrate
RTP Real-Time Transport Protocol
RTSP Real-Time Streaming Protocol
RUSBoost Random Undersampling Boosting

S

SDN Software-Defined Networking
SDP Session Description Protocol
SFU Selective Forwarding Unit

T

TCP Transmission Control Protocol

U

UDP User Data Protocol

V

VBR Variable Bitrate

VoD Video-On-Demand

VR Virtual Reality

W

WebRTC Web Real-Time Communication

Samenvatting

Videostreaming applicaties domineren momenteel het internetverkeer. Volgens rapporten van Cisco is het aandeel van videoapplicaties over het internet gestegen van 2900 Petabytes per maand in 2009 naar 58000 Petabytes per maand in 2017, een groei van maar liefst twintig keer in tien jaar. Deze groei is grotendeels te danken aan de brede toegankelijkheid van videodiensten en sterk geëvolueerde technologieën voor het afleveren van video. Websites als Facebook, Netflix en Youtube stellen een ongezien aanbod ter beschikking aan de eindgebruikers, van professionele TV-series tot niet-professionele clips. Een andere belangrijke factor is de videokwaliteit, die zowel verbeterd is door nieuwe encoderingstechnieken als door meer geavanceerde toestellen van de eindgebruikers. Al deze factoren dragen bij tot de groeiende populariteit van videodiensten.

Ook de technologieën die worden gebruikt om videodiensten af te leveren zijn doorheen de tijd sterk geëvolueerd. Oorspronkelijk was het internet niet ontworpen voor applicaties die intensief gebruik maken van de beschikbare bandbreedte, zoals videoapplicaties. Hierdoor is het optimaliseren van de gebruikerservaring, aangeduid als Quality of Experience (QoE), niet vanzelfsprekend. In Video-on-Demand (VoD) en livestreaming, zijn traditionele technologieën, gebaseerd op de RTP/RTCP protocollen, vervangen door technieken gebaseerd op het HTTP protocol. Oplossingen met HTTP zorgen voor een grotere netwerkvertraging in vergelijking met RTP maar hebben een grotere schaalbaarheid, kunnen beter overweg met NAT en firewalls en kunnen gebruik maken van de reeds bestaande HTTP-gebaseerde infrastructuur. Bovendien maakt HTTP het mogelijk om de videokwaliteit aan te passen aan veranderende netwerkcondities. De voordelen op vlak van netwerkvertraging zorgen er echter voor dat RTP/RTCP technieken, zoals Web Real-Time Communication (WebRTC), nog steeds worden gebruikt in interactieve streamingtoepassingen.

In HTTP Adaptive Streaming (HAS) wordt de video geëncodeerd volgens verschillende bitrates, genaamd kwaliteitsniveaus, en vervolgens opgesplitst in temporele segmenten. De videoclient gebruikt een kwaliteitsadaptatieheuristiek om de videokwaliteit dynamisch aan te passen aan de huidige beschikbare bandbreedte. De videospeler is uitgerust met een buffer waarin de gedownloade videosegmenten worden opgeslagen alvorens te worden afgespeeld. Mede door de adaptiviteit van HAS en door deze buffercapaciteit kunnen onderbrekingen in het afspelen van de video grotendeels worden voorkomen. Gezien de hoge populariteit van deze oplossingen, werden HAS-protocollen gestandaardiseerd door MPEG Dynamic Adaptive Streaming over HTTP (DASH) in 2012.

Veel onderzoek heeft sindsdien gestreefd naar het optimaliseren van de QoE van de eindgebruikers voor HAS-toepassingen. Een groot deel van de studies focust op het ontwerpen van nieuwe clientgebaseerde heuristieken om de kwaliteitsadaptatie van de videospelers te verbeteren. Clientgebaseerde heuristieken kunnen echter niet steeds een goede ervaring garanderen voor de eindgebruikers. Daarom presenteert het werk in deze thesis een meer geavanceerde architectuur waarbij extra intelligente componenten in het netwerk worden geplaatst om de aflevering van de video's te ondersteunen. Bovendien focussen deze componenten op het optimaliseren van specifieke QoE-parameters in plaats van Quality of Service (QoS)-parameters, die zich enkel richten op de netwerkprestaties en niet op de kijkervaring van de eindgebruikers.

Het eerste probleem dat deze thesis aanpakt, is de negatieve impact van concurrerende HAS-applicaties. Als dit probleem voorkomt, heeft de kwaliteitsadaptatie van een gebruiker een negatieve invloed op het adaptatiegedrag van de andere gebruikers. Dit gebeurt meestal wanneer deze een gemeenschappelijke netwerkverbinding delen. Dit betekent dat gebruikers met gelijkaardige netwerkcondities een totaal verschillende QoE kunnen ervaren. Hoewel een eerlijke aflevering geen echte QoE-parameter is, is het toch een belangrijke eigenschap van de videodienst. Daarom moet deze worden geoptimaliseerd omdat ze een invloed heeft op de finale QoE van elke gebruiker. Bijgevolg wordt in deze thesis een systeem van netwerkcomponenten ontwikkeld die de HAS-clients ondersteunt om hun QoE te optimaliseren en een eerlijk gedrag te hebben. Het systeem berekent een eerlijk aandeel van de bandbreedte voor elk van de concurrerende gebruikers. De clients gebruiken deze informatie in hun kwaliteitsadaptatieheuristiek. Hierdoor maximaliseren ze hun eigen QoE en houden ze rekening met de andere clients. Simulatieresultaten tonen dat de voorgestelde benadering eerlijkheid kan verbeteren tot 80% in vergelijking met de state-of-the-art heuristieken.

Een tweede manier om de QoE te optimaliseren is door het vermijden van onderbrekingen in het afspelen van de video. Deze hebben namelijk een grote negatieve invloed op de gebruikerservaring. Ook al werd HAS ontworpen om onderbrekingen te verminderen, toch komt dit veel voor bij clientgebaseerde oplossingen. Bij onbeheerde HAS-systemen hebben clients geen exacte weergave van de netwerkcondities wat resulteert in verkeerde kwaliteitsveranderingen en dus onderbrekingen. Dit probleem heeft een nog negatievere impact op livestreaming, waarbij de buffer verkleind wordt om de vertraging ten opzichte van het livesignaal te minimaliseren. Daarom stelt deze thesis een oplossing voor waarbij een netwerkcontroller, die gebaseerd is op het Software-Defined Networking (SDN) principe, gebruikt wordt. Door het toepassen van prioritisatie in het netwerk door deze controller, kan het downloaden van specifieke segmenten worden gegarandeerd. Deze beslissing wordt genomen door rekening te houden met metingen in het netwerk enerzijds en met een geschatte waarde van de buffergrootte van de client op dat moment anderzijds. Dit systeem vereist geen communicatie tussen de clients en de controller en kan elke clientimplementatie ondersteunen. In de geëvalueerde netwerkscenario's kan het ontworpen SDN-systeem onderbrekingen verminderen tot 10 keer in vergelijking met de gebenchmarkte algoritmen.

Ondanks de verbeteringen die de bestaande netwerksystemen leveren op het vlak van QoE van HAS-gebruikers, zijn deze systemen deterministisch en niet adaptief. Bovendien vereisen ze een nauwkeurige configuratie om een goede prestatie te bereiken. Zelflerende netwerkcomponenten, die de beste actie leren afhankelijk van de toestand van de streamingdienst, kunnen dit probleem verhelpen. De voordelen van deze aanpak worden aangetoond in bovenstaand prioritisatiesysteem. Hier voert een machine learning gebaseerd algoritme de prioritisatie uit. Deze is gebaseerd op het Random Undersampling Boosting (RUSBoost) algoritme en op vage logica, die autonoom leren wanneer bepaalde HAS-segmenten moeten geprioritiseerd worden. Resultaten op een emulatie testbed tonen dat de machine learning gebaseerde oplossing zowel het aantal onderbrekingen als de onderbrekingstijd respectievelijk met 65% en 45% kan reduceren, in vergelijking met een eenvoudige heuristiek zonder prioritisatie.

Een laatste optimalisatie focust op interactieve videodiensten zoals videoconferencing en afstandsleren. Zoals reeds vermeld in het begin van deze samenvatting, zijn HAS-technieken zeer effectief voor VoD en livestreaming maar staan ze niet garant voor een lage netwerkvertraging. Voor interactieve videodiensten worden daarom RTP-gebaseerde protocollen gebruikt. In het bijzonder is de WebRTC technologie hiervoor een ideale kandidaat, omdat deze niet gebruikt maakt van externe plugins en compatibel is met elk toestel dat een browser ondersteunt. WebRTC werkt echter peer-to-peer en is daarom niet schaalbaar wanneer er vele gebruikers willen deelnemen aan de sessie. De reden daarvoor is dat elke peer een aparte stream moet encoderen voor elke andere peer in de sessie. Om dit probleem van schaalbaarheid op te lossen wordt er in deze thesis een systeem ontworpen waar elke peer enkel een beperkt aantal streams moet encoderen, veel kleiner dan het aantal deelnemers. Deze streams worden verzonden naar een centrale component, genaamd de conferentiecontroller, die de streams doorstuurt naar de ontvangende peers, afhankelijk van hun beschikbare bandbreedte. Elke geëncodeerde stream wordt op deze manier verzonden naar meerdere ontvangers tegelijk, wat de schaalbaarheid vergroot. Bovendien herberekent de controller periodiek de encoderingsbitrates van de zendende peers om de bandbreedtevariaties van de ontvangers beter te volgen. Zo wordt de algemene videokwaliteit van de ontvangers verbeterd. Op een emulatie scenario met één zendende peer en 28 ontvangende peers verbetert de voorgestelde benadering de ontvangende videorate tot met 15%, in vergelijking met een oplossing waarbij de encoderingsbitrates statisch zijn en niet veranderen.

De oplossingen die worden voorgesteld in deze thesis behandelen enkele belangrijke uitdagingen voor adaptieve videodiensten door het gebruik van intelligente netwerkcomponenten die videoclients ondersteunen in het maximaliseren van hun QoE. Toekomstig onderzoek kan hier verder op bouwen door te focussen op nieuwe opportuniteiten en uitdagingen. Er zullen onder andere nieuwe streamingapplicaties opduiken die een kleine netwerkvertraging en meer bandbreedte vereisen, zoals immersive videostreaming. Ook nieuwe applicaties die gekenmerkt worden door zeer lage vertraging en hoge mobiliteit zullen aan populariteit winnen. Hierbij wordt niet enkel gedacht aan real-time applicaties, maar

ook machine-to-machine communicatie zoals beveiligingssystemen met drones. Nieuwe netwerkmodellen zoals 5G en softwaregebaseerde netwerken, die ook de bovenstaande communicatievormen ondersteunen, zullen noodzakelijk zijn om een goede gebruikerservaring te garanderen. Ondanks deze nieuwe opportuniteiten zullen er nog uitdagingen komen voor netwerkgebaseerde oplossingen. Ten eerste zal internetverkeer nog meer geëncrypteerd worden in de toekomst. Deze factor maakt het moeilijker voor netwerkgebaseerde toepassingen omdat QoE-parameters hierdoor niet toegankelijk zijn. Deze parameters zullen dus berekend moeten worden. Ten tweede zullen nieuwe transportprotocollen, zoals Quick UDP Internet Connection (QUIC), nieuwe functionaliteiten aanbieden die het afleveren van adaptieve videodiensten kunnen verbeteren.

Summary

Video streaming applications currently dominate Internet traffic. According to Cisco reports, the amount of traffic generated by video applications has increased from 2900 Petabytes per month in 2009 to 58000 Petabytes per month in 2017, an impressive growth of almost twenty times in less than ten years. This growth can be rooted back to two different causes, namely widespread content accessibility and improved delivery techniques. Websites as YouTube, Netflix and Facebook guarantee an unseen availability of video content to Internet users, ranging from professional TV series to user-generated content. Video quality has also improved consistently in recent years, both in terms of encoding techniques and user device capabilities. These aspects have made video content more appealing than ever to the end users.

This streaming revolution has also occurred thanks to relevant improvements in the video delivery techniques. Indeed, the best-effort Internet was not originally designed to support such bandwidth-intensive traffic. This aspect poses a serious challenge on how to provide a good experience to the video streaming users, the so-called Quality of Experience (QoE). In Video-on-Demand (VoD) and live streaming events, traditional streaming techniques based on RTP/RTCP have been replaced by solutions based on the HTTP protocol instead. At the cost of an increased end-to-end latency compared to RTP, HTTP-based streaming presents an increased scalability, better NAT and firewall traversal and the possibility to reuse the existing HTTP infrastructure, in terms of caches and web servers. Moreover, this technique allows to easily accommodate the streamed video quality to the varying bandwidth conditions of the video clients. Nevertheless, RTP-based techniques did not disappear, but they are actually widely used in interactive streaming scenarios, as in the Web Real-Time Communication (WebRTC) initiative. Indeed, HTTP-based techniques do not guarantee the low-latency and interactivity requirements of these applications.

In HTTP Adaptive Streaming (HAS), the video content is encoded at different bitrates, also called quality levels, and temporally segmented. The video client is equipped with a rate adaptation algorithm that dynamically decides the best quality level to download based on the locally perceived network conditions. The client is also equipped with a video buffer where the downloaded video segments are stored before being played out. This buffer is used to absorb temporary bandwidth fluctuations and, together with the aforementioned quality adaptation, provide a continuous playout.

Given the popularity of these solutions, the MPEG consortium has proposed

a standard for HAS in 2012, called Dynamic Adaptive Streaming over HTTP (MPEG-DASH). Since the standard was defined, a large body of research has investigated how to improve the users' QoE for HAS. Many studies focus on the development of new client-based heuristics, in order to improve the quality adaptation of the clients. Unfortunately, purely client-based heuristics can fail in delivering the best experience to the end users. For this reason, the work presented in this dissertation goes one step beyond traditional client-based algorithms, by developing additional nodes located inside the network to help the delivery of the video. Moreover, the proposed network components aim at optimizing specific QoE parameters that directly impact the users' viewing experience, rather than Quality of Service (QoS) parameters, which represent low-level network performance.

One of the first problems addressed in this dissertation is unfairness among competing HAS clients. When fairness problems emerge, the quality adaptations of clients streaming at the same time over the same bottleneck links negatively influence each other. This behavior entails that similar clients with similar network conditions can obtain a very different streaming experience. Even though fairness is a system-wide characteristic rather than a user perceived QoE factor, it is often a desired property of the system, because it can finally improve user experience and should therefore be maximized. Therefore, in this thesis, an in-network system of network components is deployed to support the HAS clients and allow them to obtain both a high QoE and a fair behavior. The in-network system estimates the fair bandwidth share of the competing clients. This information is used by the clients to request the video quality maximizing their own QoE, while being fair to the other users. Numerical results show that the proposed approach can improve fairness up to 80% when compared to state-of-the-art heuristics.

Another way of improving the QoE of HAS users is to avoid the occurrence of video playout interruptions, also called video freezes, which have the strongest impact on user experience. Even though the HAS principle was designed to avoid freezes, purely client-based solutions can still be affected by this problem. In unmanaged HAS solutions, the clients are not aware of the real bandwidth conditions of the network, and can therefore perform sub-optimal quality adaptation decisions. This issue is aggravated in live streaming scenarios, where the client buffer has to be reduced as much as possible to reduce the camera-to-display delay and guarantee a near-to-live experience to HAS users. To reduce this problem, this thesis proposes a network framework, based on the Software-Defined Networking (SDN) principle, that can temporarily prioritize the delivery of HAS segments possibly leading to a video freeze. This decision is based on both measurements collected from the network nodes and an estimation of the client status, in terms of video player buffer and requested quality level. The proposed framework does not require any communication between clients and network, and can optimize the delivery of any client implementation. In the evaluated emulated network scenarios, the proposed SDN system results in a reduction of video freeze time up to 10 times, when compared to the benchmarking algorithms.

Even though the proposed in-network systems can effectively improve the delivery of HAS streams, they are affected by one drawback, namely lack of adapt-

ability. More specifically, they are hardwired and require a certain degree of human fine-tuning to be able to obtain the best performance. Self-learning network nodes can overcome this problem, by learning the best action to take depending on the streaming scenario. The benefits of this approach are showcased in the prioritization framework presented above. Particularly, a machine learning-based prioritization algorithm is designed to perform the prioritization task. Prioritization is based on the Random Undersampling Boosting (RUSBoost) algorithm and fuzzy logic, which can autonomously learn when an HAS segment should be prioritized. Results obtained through emulation shows that the machine learning-based approach can consistently reduce video freezes with about 65% and freeze time with 45%, when compared to a baseline heuristic without prioritization.

A final optimization approach is presented for remote video collaboration applications. As mentioned at the beginning of this summary, HAS techniques are very effective for VoD and live streaming scenarios, but they do not guarantee the low latency required for remote collaborations. In this case, protocols based on RTP are used instead. More particularly, the WebRTC technology suite is an ideal candidate, as it is browser-based and does not require external plugins, which makes it compatible with any device. Nevertheless, WebRTC is peer-to-peer by design and therefore presents scalability problems when the number of participants to the remote session is large. Ideally, each peer would need to encode a separate stream for any other peer in the session. To improve the scalability of this peer-to-peer architecture, a framework is proposed where each peer only needs to encode a limited number of streams, much smaller than the number of participants. These streams are sent to a centralized node, called conference controller, which dynamically forwards them to the receiving peers, based on their bandwidth conditions. This way, each encoder at the sender transmits to a multitude of receivers at the same time, improving scalability. Moreover, the controller periodically recomputes the set of encoding bitrates at the sending peers, in order to follow the long-term bandwidth variations of the receivers and increase the received video quality. In an emulated scenario with a single sending peer transmitting to 28 receivers, the proposed approach increases the received video rate up to 15%, compared to a solution where the encoding bitrates are static and do not change over time.

The proposed approaches address important challenges in the delivery of adaptive video streaming services, by using intelligent network elements that can support the video clients and help them reaching an excellent QoE. Several future opportunities and challenges can also be identified both in the domain of network-assisted solutions and adaptive streaming in general, which can profit from the work presented in this dissertation. New streaming applications will appear in the near future, as immersive video streaming, which are extremely bandwidth intensive and require careful delivery in terms of quality and latency. Also, ultra-low latency and high-mobility applications will become more and more important. These scenarios concern not only real-time applications, but also machine-to-machine communication, as in car-to-car entertainment systems or drones surveillance. New network paradigms as 5G and softwarized networks, which can support both traditional forms of communication and more unstructured ones, will be

essential to provide the best service to the final users. Despite these new opportunities, there are also challenges ahead for network-assisted solutions. First, it is expected that an increasingly larger portion of Internet traffic will be encrypted in the next few years. This aspect complicates how network-assisted solutions work, since video streaming performance parameters will not be easily accessible anymore and would need to be estimated instead. Second, new transport layer protocols, as the Quick UDP Internet Connection (QUIC) protocol, provide new functionalities that can be successfully employed to optimize the delivery of adaptive video streaming services.

1 Introduction

"As soon as you have an idea that changes some small part of the world you are writing science fiction. It is always the art of the possible, never the impossible."

–Ray Bradbury (1920 – 2012)

1.1 The Streaming Revolution

April 23, 2005. Jawed Karim, one of the YouTube co-founders, uploads the first video ever on the newly born YouTube website. The video *Me at the zoo* had a duration of about 19 seconds and showed Jawed in front of some elephants at the zoo of San Diego, California. Little did he know, that single video would change the video streaming industry forever, revolutionizing the way we enjoy our free time, learn and interact among each other. Nowadays, more than one billion hours of video are watched on YouTube every day. Every minute, more than 400 hours of video are uploaded and shared.

January 15, 2007. Netflix, a company specialized in the DVD rental by mail business, launches its on-demand video streaming website to set the final offensive towards Blockbusters, the leading company in the video rental industry at the time. Netflix has now more than 100 million subscribers around the world. On the US Internet during peak hours, Netflix accounts for almost 40% of the total downstream traffic [1].

These two events have not only radically changed video streaming technologies, but have also posed a consistent strain on the Internet infrastructure itself. In-

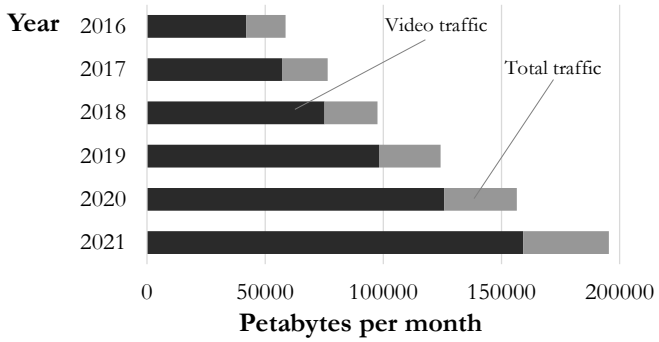


Figure 1.1: According to the Cisco Visual Networking Index [2], 82% of the Internet traffic is going to be video in 2021. Moreover, video traffic is going to increase by almost four times in the period 2016–2021.

deed, the Internet was not designed to support such real-time bandwidth-consuming applications, but was originally conceived to transport and share files and data, which are usually characterized by less stringent requirements than video. This trend is not going to stop, as video streaming traffic is going to account for almost 82% of the total Internet traffic by 2021 (Figure 1.1), according to the Cisco Visual Networking Index [2]. The streaming revolution described at the beginning of this chapter was mainly possible because of two factors: increased network speed and improvement of video streaming protocols.

Network bandwidth has steadily grown since the beginning of the Internet. When the first commercial modem was manufactured in 1962, the Bell 103 by AT&T, it had a nominal downlink speed of 300 bps. In 2015, the average connection speed in the USA was about 12.6 Mbps [3], and this value is steadily increasing around the world. As an example, according to the empirical Nielsen's law on Internet bandwidth, a high-end user's connection speed grows by 50% per year[1]. The improvement in network performance, both in terms of speed and pervasiveness, has been the ideal breeding ground for the rapid growth of services as YouTube and Netflix.

As far as streaming protocols are concerned, the scale of current streaming platforms have caused a shift from protocols based on the Real-Time Transport Protocol (RTP), to those based on HTTP. The RTP standard was designed for the real-time transport of videos over the best-effort Internet using the User Data Protocol (UDP), which was preferred over the Transmission Control Protocol (TCP) due to the low latency of UDP. RTP adds a series of optimizations on top of UDP to support video streaming (as sequence numbers, time stamps, priority indicators etc.), so that out-of-order datagrams can be re-ordered and errors can be

[1] https://www.nngroup.com/articles/law-of-bandwidth/

detected. RTP streaming allows to reach a very low end-to-end latency, which is extremely important in interactive applications (e.g., remote teaching, video conferencing etc.). Nevertheless, RTP-based solutions have been gradually replaced by approaches based on HTTP, which can reuse the highly optimized and scalable infrastructure of web servers and caches already available in the Internet, while also being compatible with firewalls and NAT-traversal. Progressive download over HTTP has been used at first to deliver videos over the best-effort Internet. With progressive download, the playback can start while the video is still being downloaded. This technology has paved the way to HTTP Adaptive Streaming (HAS), the current dominant technology for video streaming over the Internet. In HAS, the video content, either captured live or on-demand, is encoded at multiple different bitrates (also called quality levels) and chunked into small segments, each containing a few seconds of the video. A manifest file, stored at the server, specifies the structure of the video in terms of available quality levels and segments. At the beginning of the streaming session, the client downloads the manifest file from the server and starts requesting the video segments in temporal order. The video playback starts after one or more segments are stored in the client buffer, which is required to absorb temporary bandwidth fluctuations. Beside the advantages already introduced by progressive download over HTTP, HAS also allows to dynamically switch the bitrate of the video during the streaming session, in order to accommodate for bandwidth fluctuations. The switching logic, also called rate adaptation algorithm, completely resides at the client, which makes this approach highly scalable. The goal of the adaptation algorithm is to match the video bitrate to the network bandwidth, with the primary focus of providing a continuous playout while maximizing the streamed quality. In light of the wide adoption of the HAS principle in many proprietary solutions, the MPEG consortium has created a standard for HAS in 2012, called MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [4]. Standardization mainly covers the structure of the manifest of the video, while the client rate adaptation heuristic is outside MPEG-DASH's scope. Moreover, MPEG-DASH is completely codec-agnostic, meaning it is compatible with any codec format.

HAS has proven to be very effective when it comes to Video-On-Demand (VoD) and live video streaming, but it has not completely replaced RTP. HAS introduces by design several seconds of latency between the server and the client, which makes this technology unable to guarantee the low latency requirements of interactive video applications. As a general rule of thumb, video conferencing or remote teaching applications require an end-to-end latency smaller than 300 ms to be considered acceptable [5]. Even in the best case, current HAS solutions are not able to provide an end-to-end latency smaller than one/two seconds [6]. For this reason, interactive streaming applications still widely use protocols based on RTP, as the cross-browser compatible open-source Web Real-Time Communica-

tion (WebRTC). Particularly, WebRTC comes with a peer-to-peer architecture in mind and allows plugin-free real-time communication among browsers, and is being standardized by the world wide web consortium and the Internet engineering task force. Ideally, each device equipped with a browser is capable of initiating a WebRTC session, which makes it an ideal candidate for new interactive streaming applications, as telehealth or remote teaching.

1.2 Challenges

As discussed in the previous section, multimedia content now represents a large portion of the Internet traffic, and its relevance is expected to increase in the years to come. This aspect poses a serious challenge on how to efficiently deliver multimedia content over the Internet. An effective delivery depends on the quality as perceived by the end user, the so-called Quality of Experience (QoE), rather than on classical Quality of Service (QoS) network-level parameters. As the relationship between QoS parameters and QoE is far from linear, a classical QoS-centric delivery is not able to fully optimize the quality as it is perceived by the end users. The international telecommunication union defines the QoE as "*the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state*" [7]. As stated above, the concept of QoE not only includes application-specific factors that impact the viewing experience, but it also incorporates the context, personality and state of the user watching the video. These user-related factors can only be taken into account by performing subjective studies, an important and very active research field in the broader domain of QoE optimization [8]. Despite their importance, subjective studies are unfortunately time- and resource-consuming. For example, the QoE model employed in Chapter 3 has been obtained via a crowdsourcing study involving almost 500 volunteers [9]. The goal of this thesis is therefore to analyze the problem of QoE-centric video delivery from an aggregate service degradation point of view, rather than evaluating the individual perception of each user in the system, and particularly to design those control actions that can be taken to optimize the video delivery. For this reason, this thesis focuses on optimizing those general QoE factors, as video quality and freezes for instance, that are clearly quantifiable and have a direct impact on user experience in video streaming.

Moreover, the work presented in this PhD thesis makes use of network-assisted elements, with the aim of supporting the delivery of adaptive video streams and providing a better QoE to the end users. This shift is needed as several QoE factors in video streaming cannot be optimized using a purely client-server architecture, but would benefit from the assistance of intelligent network elements. As

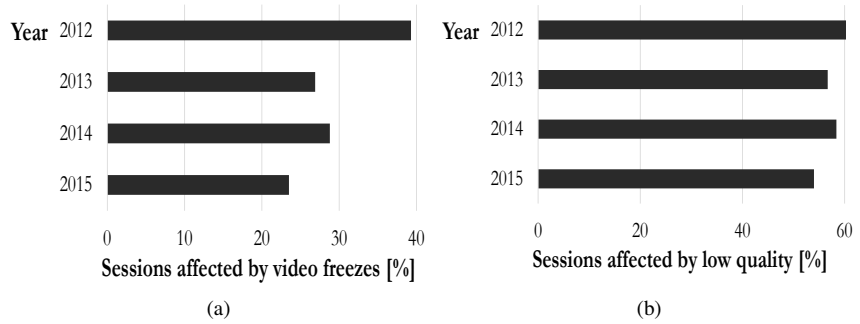


Figure 1.2: Rebuffering events (1.2a) and low quality (1.2b) are still common in nowadays Internet video streaming [10].

an example, despite the bandwidth adaptation capabilities of HAS clients, current HAS solutions can still suffer from video freezes and low video quality [10], as shown in Figure 1.2. This problem is mainly due to the unmanaged nature of current streaming technologies, as the clients are only aware of the local perceived bandwidth conditions and cannot be assisted in improving the delivered QoE. Delivering a sub-optimal experience is known to have a negative influence on user engagement and, therefore, on the profitability of the video streaming service. As an example, it has been reported that most of the users will abandon the streaming session in less than 60 seconds if the video is affected by low quality or video freezes [11]. The network elements presented in this thesis are therefore developed to assist and support the delivery of the video, and help the clients obtaining good streaming performance. In light of the above, the following challenges are identified and addressed in this thesis:

Challenge #1: provide a video streaming service that is fair to the end user from the QoE point of view. In a typical HAS setting, multiple clients can access the same content from the same server. Often, clients have to share a single medium and issues concerning fairness among them appear, meaning that the presence of a client has a negative impact on the performance of others. This behavior particularly emerges when multiple HAS clients have to share one or multiple bottleneck links. As reported by Akhshabi et al. [12], fairness issues are not due to TCP dynamics, but arise from the rate adaptation algorithms, as they decide on the actual rate to download. When multiple clients stream a video at the same time, a wrong bandwidth estimation can occur due to the temporal overlap of the activity-inactivity periods of the clients. This wrong estimation subsequently affects the bitrate selection and thus the clients' QoE. The problem is aggravated by the uncoordinated nature of current HAS algorithms, which entails they are not aware of the presence of other clients nor can they adapt their behavior to deal with it.

Challenge #2: avoid playout interruptions, also called freezes, during the video playback. Even though HAS solutions have been developed with bandwidth adaptation capabilities in mind, they can still fail in avoiding video freezes. Video freezes are known to have the worst influence on user experience, and should therefore be reduced as much as possible. As reported by Akshabi et al. and Riiser et al. [13, 14], current rate adaptation heuristics perform quality selection sub-optimally, especially when a sudden bandwidth drop occurs. This leads to unnecessary quality switches and video playout interruptions, which negatively affect the final QoE of the users. Similar conclusions are drawn in the 2015 Conviva report on HAS [10]. The report reveals that almost 25% of the analyzed HAS sessions exhibit at least one video freeze. This problem is mainly due to the unmanaged nature of current HAS technologies. The clients are not aware of the real network conditions nor are they assisted in improving the delivered QoE.

Challenge #3: increase the adaptability of the network elements to support unseen video streaming scenarios, reducing human hand-tuning. Given the popularity and possible benefits of network-based solutions, the MPEG consortium has recently standardized an extension of the MPEG-DASH standard, called MPEG Server and Network-Assisted DASH (MPEG-SAND). The MPEG-SAND standard defines the interfaces and messages that can be exchanged among network elements, servers and video clients. As an example, MPEG-DASH clients can report QoE metrics that can be used by the network for monitoring purposes and/or to implement QoE-aware optimizations. The optimization taking place in the network is therefore out of the standard and can be developed based on the particular goal of the streaming provider. Even though network elements can help the video delivery, they are usually designed and developed with certain network and streaming characteristics in mind. This aspect entails that these systems might be unable to adapt and provide satisfactory performance under highly different scenarios, in terms of available bandwidth, HAS clients configurations, network topology and streamed videos. It is therefore required to make the network elements adaptive and self-learning, so that they can learn the best action to take based on the network and streaming conditions.

Challenge #4: reduce costs and increase scalability and performance for interactive streaming applications. Even though HAS solutions can efficiently stream videos over the best-effort Internet, they still suffer from a design problem, namely end-to-end latency. Indeed, HAS solutions usually rely on a video player buffer size of a few seconds at client-side, which introduces a non-negligible latency between the event captured at the server and its playout on the client's display. Low-latency streaming applications, as immersive streaming, gaming and real-time communication, will become dominant in future years. These applications are usually characterized by a high degree of interactivity and the need for low latency, which classical HAS solutions cannot always guarantee. On the other

hand, remote conferencing solutions, as those provided by WebRTC, can be used to implement such scenarios. The WebRTC framework has been developed with a peer-to-peer architecture in mind, where a small group of clients can directly communicate with each other. Unfortunately, this approach can suffer from scalability issues when multiple participants are present at the same time, since the WebRTC senders would need to encode a separate stream for each of the WebRTC receivers. This aspect entails that each receiver is associated with an independent and dedicated encoder at sender-side, which is expensive and does not scale well. To make WebRTC work in a wide range of interactive streaming scenarios, it is therefore needed to solve this scalability issue, without negatively impacting the delivered video quality.

1.3 Outline

This dissertation is composed of several publications that have been realized within the scope of this PhD. The selected publications provide an integral and consistent overview of the work performed. The different research contributions are detailed in Section 1.4 and the complete list of publications that resulted from this work is presented in Section 1.5. This section gives an overview of the remainder of this dissertation and explains how the different chapters are linked together.

A high-level introduction on the status of HTTP adaptive video streaming services is given in Chapter 2. Particularly, the presented analysis focuses on works that go beyond purely client-based solutions, with the goal of fully optimizing users' QoE. This shift is needed as several important factors influencing the user experience cannot be fully optimized by only relying on the client heuristic. Particularly, these works are characterized in three groups, based on where the optimization takes place. First, server- and network-assisted solutions place additional intelligence in the network to support the delivery of the video. Traffic rerouting, bandwidth shaping techniques and caching represent the most typical examples in this space. The works presented in this PhD thesis belong to this category. Second, application level solutions can optimize the behavior of any adaptation heuristic by exploiting, for instance, the new features of the HTTP/2 protocol or prefetching techniques. Third, transport level approaches modify the congestion control algorithms or retransmission policies of TCP, to better support video traffic. All these different solutions are also analyzed in terms of QoE benefits (e.g., quality, freezes, fairness etc.) and deployment complexity.

After this introduction, we tackle one of the issues still affecting HAS in Chapter 3, namely fairness among competing clients (*Challenge #1*). To solve this problem, a new HAS heuristic called Fair In-Network Enhanced Adaptive Streaming (FINEAS) is presented, which is able to select the best quality to request depending on the network conditions, in order to provide a smooth video

streaming experience and improve fairness. Particularly, this heuristic is able to increase the average requested quality level compared to current HAS heuristics and avoid video freezes, while guaranteeing similar QoE to the all the clients streaming video, i.e., fairness. The FINEAS heuristic is supported by an in-network-based system to help clients coordinate their behavior, which does not require explicit client-to-client communication or a centralized decision process. Consequently, the quality level selection can still be performed locally and independently by each client, without any modification to the general HAS principle. Numerical simulations using NS-3 show that the multi-client HAS framework results in a better video quality and in a remarkable improvement of fairness, up to 20% and 80% respectively, when compared to state-of-the-art heuristics.

Chapter 4 analyses an in-network solution to reduce the occurrence of video freezes, which represent the most important factor influencing user experience in video streaming services (*Challenge #2*). Despite the bandwidth adaptation capabilities of HAS, video freezes are still common, especially when sudden bandwidth drops occur. Therefore, Chapter 4 presents a Software-Defined Networking (SDN)-based framework to help the clients in avoiding video freezes caused by network congestion. The main element of this framework is a controller, which has the fundamental role of prioritizing the delivery of particular HAS segments in order to avoid video freezes. This decision is based on the HAS clients' status and on measurement data collected from the network nodes. Moreover, the HAS clients' conditions are estimated at the controller-side without any explicit client-to-controller communication. Consequently, no extra signaling overhead is introduced into the network. The proposed SDN framework is implemented on the Mininet Network Emulator, to provide realistic results under diverse network conditions. In the evaluated network scenarios, the proposed SDN framework results in a reduction of video freeze time up to 10 times, when compared to the benchmarking algorithms.

Even though the in-network approach presented in Chapter 4 shows excellent results, it is affected by two main drawbacks. First, the controller logic needs to know the characteristics of the streamed video, in terms of segment duration and bitrates of the different quality levels. Second, it requires an estimation of the buffer filling level of the clients' video player. This in turn entails that the controller has to know the initial buffering time of the client (i.e., the amount of video buffered before the playout can start). As no extra signaling is foreseen between the clients and the controller, this information can only be obtained by intercepting the manifest file requested by a client before starting the video. This aspect limits the general applicability of the proposed framework in a real environment. For this reason, Chapter 5 presents a complete re-design of the controller logic using a machine learning approach (*Challenge #3*). A freeze prediction algorithm located at the controller is able to detect beforehand when a client is going to freeze and

drive the network prioritization. The freeze prediction algorithm is trained to correlate information such as the network bandwidth and the timing of consecutive segment requests issued by a client, to the occurrence of a freeze. Consequently, the prediction does not require any a priori assumption on the video characteristics or the buffer behavior of the clients. The machine learning engine embedded in the SDN controller is based on the Random Undersampling Boosting (RUSBoost) algorithm [15] and fuzzy logic. Results obtained through emulation shows that the machine learning-based approach can consistently reduce video freezes with about 65% and freeze time with 45%, when compared to the benchmarking heuristics.

As explained in Section 1.2, HAS solutions are usually characterized by end-to-end latencies that are not compatible with interactive streaming scenarios, as for example in a remote teaching class. Real-time communication protocols, as WebRTC, can provide the required latency and degree of interactivity, but are usually not scalable due to their peer-to-peer nature. To reduce this issue, Chapter 6 presents a WebRTC-compliant framework to support the delivery of real-time communication streams, with a remote teaching scenario in mind (*Challenge #4*). In this framework, the WebRTC sender only needs to encode a limited number of streams, much smaller than the number of receivers, at different bitrates. This approach allows to overcome the aforementioned limitation, where each receiver would need to be associated to an independent, dedicated encoder. In the proposed framework instead, multiple receivers are assigned to the same encoder at sender-side. A centralized controller is aware of the bandwidth conditions of the WebRTC receivers and dynamically forwards the stream at the best bitrate in order to follow the bandwidth variations of the receivers. Besides this dynamic stream forwarding, the centralized node has another fundamental task. Instead of keeping the encoding bitrates of the sender fixed at predefined static values, the controller can dynamically and periodically recompute them based on the changing bandwidth conditions of the receivers. This approach allows to better follow the bandwidth characteristics of the receivers, even though only a limited number of encoders is actually used. An emulation testbed is also presented to test the performance of the proposed framework using state-of-the-art software. In an emulated scenario with 28 receivers and three encoders at sender-side, the proposed framework improves the average received video bitrate up to 15%, compared to a static solution where the encoding bitrates do not change over time. Moreover, the dynamic recomputation is more cost-efficient than a static approach, as less encoders are needed to obtain similar performance.

1.4 Research Contributions

In Section 1.2, the problems and challenges for the efficient delivery of adaptive streaming services are formulated. They are tackled in the remainder of this PhD

dissertation for which the outline is given in Section 1.3. To conclude, an elaborated list of the research contributions within this dissertation is presented:

- An in-network framework for the fair delivery of adaptive video streams, coupled with a client-based HAS heuristics. (Chapter 3, addressing *Challenge #1*)
  - An HAS heuristic able to select the best video quality depending on the network conditions, in order to provide a smooth video streaming experience and improve fairness from a QoE point of view.
  - An in-network-based system to help the clients coordinate their behavior, which does not require explicit client-to-client communication or a centralized decision process.
  - Extension of an NS-3 simulator to implement the proposed client-based heuristic and in-network system.
  - An extensive set of simulation results that thoroughly show the benefits of the proposed approach under diverse network conditions, client configuration, number and type of clients and the robustness towards network elements failure.
- An SDN-based framework to avoid the occurrence of video freezes at the clients. (Chapter 4, addressing *Challenge #2*)
  - An SDN controller that helps the HAS clients avoiding video freezes caused by network congestion. We assume that congestion occurs in the edge or aggregation network, where alternative routing is unavailable, and handle it using a prioritized queue.
  - An algorithm, deployed at the controller, to estimate the clients' conditions without any explicit client-to-controller communication. Consequently, no extra signaling or overhead is introduced into the network.
  - A prioritization algorithm, executed by the controller each time a client requests a video segment, that decides whether the requested video segment should be prioritized or not, based on the network and clients' conditions.
  - An emulation testbed to test the proposed framework under realistic network scenarios, which is based on the OpenFlow protocol and the MPEG-DASH standard.
  - Detailed experimental results to characterize the gains of the proposed framework compared to state-of-the-art HAS solutions, in presence of realistic Internet cross-traffic. A scalability analysis of the proposed framework shows that it can control up to several thousand video clients at the same time.

- A freeze predictor module based on machine learning to enhance the performance of the SDN controller presented above. (Chapter 5, addressing *Challenge #3*)
  - A freeze predictor based on the RUSBoost algorithm, designed to detect conditions possibly leading to a freeze at the client. The freeze predictor does not require any a priori knowledge on the characteristics of the video nor on the client's configuration.
  - A congestion detection module based on fuzzy logic, used by the controller to avoid congesting the prioritization queue and to allow a fair share among the different clients.
  - An extensive evaluation of the off-line performance of the freeze predictor, based on a dataset composed of almost 1.5 million individual video segment requests.
  - Emulation results to test the proposed machine-learning approach under realistic network conditions, which show its robustness when different videos, HAS client heuristics and network topologies are used.
- A WebRTC-based framework for a low-latency, scalable, interactive video streaming conferencing solution. (Chapter 6, addressing *Challenge #4*)
  - A centralized controller that can reduce the scalability issue of classical interactive peer-to-peer architectures, which are composed of a set of senders transmitting to a set of receivers. In this framework, the senders only need to encode a limited number of streams, much smaller than the number of receivers.
  - An algorithm for the periodical recomputation of the encoding bitrates, running at the centralized controller, to better follow the bandwidth conditions of the receivers, formulated as an integer linear programming problem.
  - A Java-based simulator to test the performance of the proposed framework under a variety of network and client conditions, and an emulation testbed based on state-of-the-art WebRTC software.
  - Extensive experiments, both in simulation and emulation, to show how the proposed framework can increase the scalability of interactive video streaming applications, the video quality delivered to the clients and make more efficient usage of the encoding resources.

1.5 Publications

The research results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. Moreover,

the work on fair adaptive streaming resulted in two international patents. The following list provides an overview of the publications and patents during my PhD research.

1.5.1 A1: Journal Publications Indexed by the ISI Web of Science "Science Citation Index Expanded"

1. **Stefano Petrangeli**, Jeroen Famaey, Maxim Claeys, Steven Latré and Filip De Turck. *QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming*. Published in ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMM), November 2015.

2. **Stefano Petrangeli**, Tim Wauters, Rafael Huysegems, Tom Bostoen and Filip De Turck. *Software-Defined Network-Based Prioritization to Avoid Video Freezes in HTTP Adaptive Streaming*. Published in International Journal of Network Management (IJNM), July 2016.

3. Jeroen van der Hooft, **Stefano Petrangeli**, Tim Wauters, Rafael Huysegems, Patrice Alface, Tom Bostoen and Filip De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks*. Published in IEEE Communications Letters (IEEE CL), August 2016.

4. Tingyao Wu, **Stefano Petrangeli**, Rafael Huysegems, Tom Bostoen and Filip De Turck. *Network-Based Video Freeze Detection and Prediction in HTTP Adaptive Streaming*. Published in Computer Communications (COMCOM), February 2017.

5. Jeroen van der Hooft, **Stefano Petrangeli**, Tim Wauters, Rafael Huysegems, Tom Bostoen and Filip De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Published in Journal of Network and Systems Management (JNSM), March 2017.

6. **Stefano Petrangeli**, Tingyao Wu, Tim Wauters, Rafael Huysegems, Tom Bostoen and Filip De Turck. *A Machine Learning-Based Framework for Preventing Video Freezes in HTTP Adaptive Streaming*. Published in Journal of Network and Computer Applications (JNCA), September 2017.

7. **Stefano Petrangeli**, Jeroen van der Hooft, Tim Wauters, and Filip De Turck. *Quality of Experience-Centric Management of Adaptive Video Streaming Services: Status and Challenges*. Published in ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMM), March 2018.

8. **Stefano Petrangeli**, Dries Pauwels, Matúš Žiak, Jeroen van der Hooft, Jürgen Slowack, Tim Wauters and Filip De Turck. *A Scalable WebRTC*

Framework for Remote Video Collaboration Applications. Submitted to Multimedia Tools and Applications, December 2017.

1.5.2 P1: Proceedings Included in the ISI Web of Science "Conference Proceedings Citation Index – Science"

1. Alessandro Verdolini and **Stefano Petrangeli**. *A Smartphone Agent for QoE Evaluation and User Classification over Mobile Networks*. In proceedings of the International Workshop on Quality of Multimedia Experience (QoMEX), July 2013.

2. **Stefano Petrangeli**, Maxim Claeys, Steven Latré, Jeroen Famaey and Filip De Turck. *A Multi-Agent Q-Learning-Based Framework for Achieving Fairness in HTTP Adaptive Streaming*. In proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), May 2014.

3. Jeroen van der Hooft, **Stefano Petrangeli**, Maxim Claeys, Jeroen Famaey and Filip De Turck. *A Learning-Based Algorithm for Improved Bandwidth Awareness of Adaptive Streaming Clients*. In proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM), May 2015.

4. **Stefano Petrangeli**, Niels Bouten, Emanuel Dejonghe, Jeroen Famaey, Philip Leroux and Filip De Turck. *Design and Evaluation of a DASH-Compliant Second Screen Video Player for Live Events in Mobile Scenarios*. In proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM), May 2015.

5. **Stefano Petrangeli**, Tim Wauters, Rafael Huysegems, Tom Bostoen and Filip De Turck. *Network-Based Dynamic Prioritization of HTTP Adaptive Streams to Avoid Video Freezes*. In proceedings of the IFIP/IEEE International Workshop on Quality of Experience Centric Management (QCMan), May 2015.

6. **Stefano Petrangeli** and Filip De Turck. *QoE-Centric Management of Advanced Multimedia Services*. In proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS), June 2015.

7. **Stefano Petrangeli**, Niels Bouten, Maxim Claeys and Filip De Turck. *Towards SVC-Based Adaptive Streaming in Information Centric Networks*. In proceedings of the Workshop on Multimedia Streaming in Information-Centric Networks (MuSIC), July 2015.

1.5.3 C1: Other Publications in International Conferences

1. Rafael Huysegems, Jeroen van der Hooft, Tom Bostoen, Patrice Rondao Alface, **Stefano Petrangeli**, Tim Wauters and Filip De Turck. *HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming*. In proceedings of the ACM Multimedia Conference (MM), October 2015.

2. Jeroen van der Hooft, **Stefano Petrangeli**, Niels Bouten, Tim Wauters, Rafael Huysegems, Tom Bostoen and Filip De Turck. *An HTTP/2 Push-Based Approach for SVC Adaptive Streaming*. In proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2016.

3. **Stefano Petrangeli**, Jeroen van der Hooft, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen and Filip De Turck. *Live Streaming of 4K Ultra-High Definition Video over the Internet*. In proceedings of the ACM Multimedia Systems (MMSys) Conference, May 2016.

4. **Stefano Petrangeli**, Patrick Van Staey, Maxim Claeys, Tim Wauters and Filip De Turck. *Energy-Aware Quality Adaptation for Mobile Video Streaming*. In proceedings of the International Conference on Network and Service Management (CNSM), November 2016.

5. Dries Pauwels, Jeroen van der Hooft, **Stefano Petrangeli**, Tim Wauters, Danny De Vleeschauwer and Filip De Turck. *A Web-Based Framework for Fast Synchronization of Live Video Players*. In proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), May 2017.

6. Jeroen van der Hooft, **Stefano Petrangeli**, Tim Wauters, Rameez Rahman, Nico Verzijp, Rafael Huysegems, Tom Bostoen and Filip De Turck. *Analysis of a Large Multimedia-Rich Web Portal for the Validation of Personal Delivery Networks*. In proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), May 2017.

7. **Stefano Petrangeli**, Viswanathan Swaminathan, Mohammad Hosseini and Filip De Turck. *Improving Virtual Reality Streaming using HTTP/2*. In proceedings of the ACM Multimedia Systems (MMSys) Conference, June 2017.

8. **Stefano Petrangeli**, Viswanathan Swaminathan, Mohammad Hosseini and Filip De Turck. *An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos*. In proceedings of the ACM Multimedia (MM) Conference, October 2017.

9. Jeroen van der Hooft, Cedric De Boom, **Stefano Petrangeli**, Tim Wauters and Filip De Turck. *An HTTP/2 Push-Based Framework for Low-Latency Adaptive Streaming Through User Profiling*. In proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2018.

10. **Stefano Petrangeli**, Dries Pauwels, Jeroen van der Hooft, Jürgen Slowack, Tim Wauters and Filip De Turck. *Dynamic Video Bitrate Adaptation for WebRTC-Based Remote Teaching Applications*. In proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2018.

1.5.4 Patents

1. **Stefano Petrangeli**, Jeroen Famaey and Steven Latré. *Fair HAS Streaming*. US patent 9826016, Accepted on 21 November 2017.

2. **Stefano Petrangeli** and Jeroen Famaey. *Fair HAS Streaming*. European patent application EP15156235.2, Submitted on 24 February 2015.

References

[1] Sandvine. *Exposing the Technical and Commercial Factors Underlying Internet Quality of Experience*. https://www.sandvine.com/trends/global-internet-phenomena/, 2016.

[2] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf, 2016.

[3] Akamai Technologies. *State of the Internet*. https://www.akamai.com/us/en/multimedia/documents/report/q3-2015-soti-connectivity-final.pdf, 2015.

[4] I. Sodagar. *The MPEG-DASH Standard for Multimedia Streaming Over the Internet*. IEEE MultiMedia, 18(4):62–67, April 2011.

[5] International Telecommunication Union. *International Telephone Connections and Circuits - General Recommendations on the Transmission Quality for an Entire International Telephone Connection*. http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=6254, 2003.

[6] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Journal of Network and Systems Management, pages 1–28, 2017.

[7] International Telecommunication Union. *Recommendation P.10: Vocabulary for performance and quality of service, Amendment 5*. https://www.itu.int/rec/T-REC-P.10, 2016.

[8] M. N. Garcia, F. D. Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnström, and A. Raake. *Quality of Experience and HTTP Adaptive Streaming: A Review of Subjective Studies*. In 2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX), pages 141–146, Sept 2014.

[9] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for estimating QoE of video delivered using HTTP adaptive streaming*. In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 1288–1293, May 2013.

[10] CONVIVA. *Mid-Year 2015 Update: Conviva Viewer Experience Report*. http://www.conviva.com/conviva-viewer-experience-report/midyear-vxr-2015/, 2015.

[11] CONVIVA. *Don't Break the Spell: a 2015 UK Consumer Survey Report*. https://www.conviva.com/research/2015-uk-consumer-survey-report-dont-break-the-spell/, 2015.

[12] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. *What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?* In Proceedings of the 22Nd International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV '12, pages 9–14. ACM, 2012.

[13] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptive Video Players over HTTP*. Image Commun., 27(4):271–287, April 2012.

[14] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. *A Comparison of Quality Scheduling in Commercial Adaptive HTTP Streaming Solutions on a 3G Network*. In Proceedings of the 4th Workshop on Mobile Video, pages 25–30, NY, USA, 2012. ACM.

[15] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. *RUSBoost: A Hybrid Approach to Alleviating Class Imbalance*. Trans. Sys. Man Cyber. Part A, 40(1):185–197, January 2010.

2

QoE-Centric Management of Adaptive Video Streaming Services: Status and Challenges

S. Petrangeli, J. van der Hooft, T. Wauter and F. De Turck.

Published in ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), March 2018.

⋆ ⋆ ⋆

The second chapter of this dissertation presents a general introduction of the HTTP Adaptive Streaming (HAS) principle and discusses the current status of HAS solutions. Particularly, the chapter focuses on solutions that go beyond purely client-based approaches that, despite effective, can still fail providing the end users with a high Quality of Experience (QoE). An elaborated survey of the current research efforts in HAS is presented, by clustering the different approaches based on where the optimization takes place: (i) server- and network-assisted solutions, (ii) application level optimizations and (iii) transport level modifications. Moreover, a set of recommendations is given to identify which QoE factor (e.g., video quality, freezes etc.) can be better improved by each solution, and its deployment complexity.

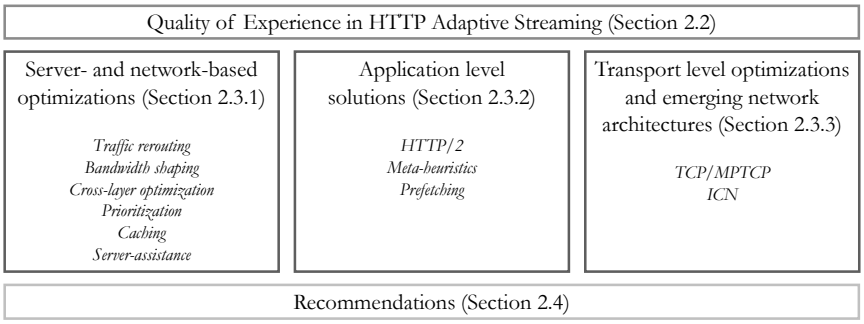


Figure 2.1: Starting from QoE aspects in HAS, we then review the status of QoE-centric management solutions and present guidelines and recommendations on the aforementioned solutions.

2.1 Introduction

Internet traffic is currently dominated by video streaming applications. Video traffic is expected to grow from 42 Exabytes per month in 2016 to 159 in 2021, an impressive 279% growth rate [1]. One of the success factors for this diffusion is the wide adoption of the HTTP adaptive streaming (HAS) principle, which has gradually replaced traditional delivery techniques using the RTP/RTSP protocol suite and progressive download. In HAS, the video is encoded at different quality levels and temporally segmented, so that each segment is an independent object or file. The client is informed of the characteristics of the video via a Media Presentation Description (MPD), which describes the available bitrates and quality levels, among others. A rate adaptation heuristic, deployed at the client, dynamically decides the bitrate of each segment to download, based on the buffer status and the perceived network conditions. The goal of the heuristic is to match the video bitrate to the network bandwidth, with the primary focus of providing a continuous playout while maximizing the streamed quality. Being based on the HTTP protocol, this approach allows for an easy deployment and firewall traversing.

In light of the wide adoption of the HAS principle in many proprietary solutions, the MPEG consortium has created a standard for HAS in 2012, called Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [2]. Standardization mainly covers the structure of the MPD of the video, while the client rate adaptation heuristic is outside MPEG-DASH's scope. Moreover, MPEG-DASH is completely codec-agnostic, meaning it is compatible with any codec format.

Since the standard was defined, a large body of research has investigated how to improve users' Quality of Experience (QoE) for HAS [3–5]. Many studies focus on the development of new heuristics, in order to improve the quality adaptation of the client [6–8]. In recent years though, several approaches have been pro-

posed that go beyond traditional client-based algorithms, with the goal of fully optimizing users' QoE [9–11]. This shift was needed as several QoE aspects in HAS cannot be completely optimized by the heuristic itself. For instance, despite the capability to adapt to varying network conditions, current HAS solutions can still suffer from video freezes. The 2015 Conviva report shows that almost 25% of the analyzed HAS sessions are affected by at least one freeze [12]. The same report also highlights that low video quality still impacts 54% of the sessions. This problem is mainly due to the unmanaged nature of current HAS technologies, as the clients are only aware of the local perceived bandwidth conditions and cannot be assisted in improving the delivered QoE. Moreover, the occurrence of freezes becomes more prominent during live streaming sessions, where the video player buffer has to be reduced as much as possible in order to minimize the live latency. In current HAS deployments, this latency is in the order of tens of seconds, because a large buffer at the client is usually required to minimize playout freezes and the server only sends a new video segment once a request is issued by the client. A purely client-based solution can also perform sub-optimally when multiple streaming clients compete for shared bandwidth [13]. Fairness issues are not due to TCP dynamics, but mainly arise from the rate adaptation algorithms, as they decide on the actual rate to download. When multiple clients retrieve video at the same time, wrong bandwidth estimations can occur, due to the temporal overlap of the activity-inactivity periods of different clients. This wrong estimation subsequently affects the bitrate selection and thus the final QoE. Previous examples identify scenarios where purely client-based solutions are not able to guarantee the best QoE to the users. Consequently, this chapter surveys existing works that optimize the delivery architecture of classic HAS systems, beyond client-based heuristics. These approaches can be categorized into three groups, based on where the optimization takes place: (1) server- and network-based approaches, (2) application level optimizations and (3) transport- and network-layer modifications.

The contributions of this chapter are twofold. First, we provide an overview on the status of existing works in the aforementioned three areas. The analysis presented in this chapter complements previous surveys on HAS, which mostly focus on existing rate adaptation heuristics only [3–5]. Second, we provide guidelines on the best strategy to adopt depending on the QoE factor to be optimized and the deployment complexity. These guidelines can be helpful for future researchers in the adaptive video streaming domain.

The remainder of this chapter is structured as depicted in Figure 2.1. Section 2.2 introduces the main elements of the HAS principle. Particularly, Section 2.2.1 describes the general architecture of adaptive streaming services, while Section 2.2.2 describes the main factors influencing QoE in HAS. Section 2.3 reports the current status of QoE-centric management of HAS services, reviewing existing works and clustering them in three different groups: server- and network-based

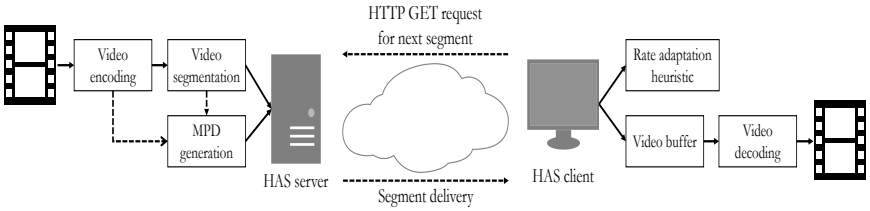


Figure 2.2: In HAS, a server hosts the video, which is encoded at different qualities and segmented. A rate adaptation heuristic, deployed at the client, determines the quality level to be downloaded.

solutions (Section 2.3.1), application level optimizations (Section 2.3.2) and transport level modifications (Section 2.3.3). In order to facilitate future research in this domain, Section 2.4 provides general guidelines on the best solution to adopt based on the network and streaming scenario. Section 2.5 concludes the chapter.

2.2 The HTTP Adaptive Streaming Principle

2.2.1 Architecture and Components

A traditional HAS architecture is composed of two elements: a server hosting the MPD of the video and the content itself, and a client streaming the video using the HTTP protocol (Figure 2.2). At the server-side, the content is encoded at different quality levels and temporally segmented, each segment usually being 1 to 10 seconds long, depending on the deployment. When selecting the segment duration for the video, two aspects have to be balanced, namely encoding efficiency and network adaptability. Short segments guarantee a fine-grained bandwidth adaptability, but they also increase the network overhead, in terms of GET requests that should be issued to the server, and are less efficient in terms of encoding. Long segments provide better performance in terms of encoding efficiency and network overhead, but they reduce the bandwidth adaptation capability of the video client.

In HAS, each quality level can be decoded independently from the others and is characterized by a specific encoding bitrate, resolution, framerate and codec [14]. When layered encoding is used instead, higher quality levels can only be decoded in combination with lower layers [15]. This approach reduces the storage requirements for the video at the cost of an increased encoding overhead per layer. The actual encoding bitrates of the video can be decided based on the characteristics of the devices accessing the video stream (e.g., smartphone, table, 4K television etc.) and on the most likely network conditions (e.g., 3G, 4G, fixed network etc.), as proposed for example by Apple HLS [16]. Another option is to made the video encoding bitrates content-specific, as proposed by Netflix [17]. In this case, each

specific video content is associated to a particular set of encoding bitrates, which are determined taking into account the characteristics of the video itself.

In order to start a streaming session, the client first downloads and parses the MPD of the video to retrieve information on the available quality levels. A rate adaptation heuristic dynamically decides the best quality level to download, for example based on network conditions and device capabilities. Once the right quality has been selected, an HTTP GET request is issued to the server to download the segment. This selection is repeated periodically, after each segment has been downloaded. The client is also equipped with a buffer where segments are stored before being played, which is used to absorb temporary bandwidth fluctuations and avoid video freezes as much as possible.

This general architecture is usually valid for any HAS deployment. To guarantee sufficient interoperability among different devices and video servers, the MPEG consortium has created the MPEG-DASH standard in 2012. This standard mainly covers the structure of the MPD file describing the video content. An MPEG-DASH MPD file is composed of one or more *periods*, where each period indicates a part of the streamed event. Each period is composed of one or multiple *adaptation sets*, which contain information about the different encoded alternatives for a specific media content (e.g., video or audio). These alternatives are called *representations* and are characterized by different bitrates, resolution, codec. Finally, each representation is composed of multiple *segments*, which are downloaded by the client to actually stream the video content. Besides defining the MPD structure, the standard allows seamless ad insertion and streaming from multiple CDNs and servers, while also defining a set of quality metrics that the HAS clients can use to report the performance of the streaming session back to a monitoring server. It is worth stressing that the heuristic employed by the client to perform the quality adaptation is not in the scope of the MPEG-DASH standard. For this reason, several client-based heuristics have been proposed in literature after the standard was defined.

Adaptation heuristics can be roughly divided into three groups: throughput-based, buffer-based and hybrid. Throughput-based heuristics mostly rely on the estimation of the available bandwidth performed by the client to select the bitrate of the next segment. The CS2P algorithm by Sun et al. falls in this category [6], for example. The authors improve the throughput prediction of HAS clients by developing a prediction model based on past video sessions, which is built offline in a node located in the streaming service provider network. This model is then used online by the clients, which remain the sole responsible of the actual quality adaptation. Buffer-based heuristics only use the buffer filling level to make a decision on the quality to request. The BBA heuristic from Huang et al. defines three operational regions based on the buffer occupancy *B*: reservoir ($B < B_{min}$), upper reservoir ($B > B_{max}$) and cushion ($B_{min} < B < B_{max}$) [7]. In the

reservoir and upper reservoir regions, the client requests the lowest and highest bitrate, respectively. In the cushion region, a monotonically increasing function is defined to select the bitrate based on the buffer. Another buffer-based heuristic is the BOLA algorithm proposed by Spiteri et al. [18]. The authors formulate the adaptation process as a utility maximization problem, where the utility depends on the achieved quality and number of video freezes. An online control algorithm is developed using the Lyapunov optimization, which guarantees the achieved utility is within a certain limit of the optimal off-line solution. Hybrid approaches combine both throughput and buffer measurements in the adaptation process, as in the PANDA case [8]. The PANDA client takes inspiration from the TCP congestion control. A target data rate is set by the client to probe the network; the requested bitrate and the time interval between consequent requests are computed to match this target, by also keeping into account the buffer filling level to reduce freezes. The hybrid SQUAD algorithm selects the next quality by primarily minimizing quality switches [19]. A feasible set of possible qualities is created at each decision step by considering only qualities whose expected download time is lower than the segment duration. This constraint is relaxed when the buffer level is above a given threshold.

We refer to the works by Seufert et al., Kua et al. and Sani et al. for an exhaustive discussion on HAS rate adaptation heuristics [3–5]. As pointed out in Section 2.1, several QoE factors (as freezes, live latency or fairness) cannot be completely optimized by classical client-based heuristics, which therefore need to be supported by optimizations taking place in the network, at the application or at the transport level. Consequently, this chapter mainly analyzes works falling in these three categories, as reported in Section 2.3.

2.2.2 QoE Factors in HAS

The ultimate goal of any streaming solution is to provide a good QoE to the end users of the service. Even though a general QoE model for HAS has not been developed yet, it is still possible to identify crucial factors having a strong influence on the QoE of adaptive streaming users, which are discussed in the remainder of this section. The same QoE factors are also used to discuss and compare the different QoE-centric solutions presented in Sections 2.3 and 2.4.

Video freezes The first and foremost objective is to avoid video freezes, which have the strongest impact on user experience. Even though the HAS principle was designed to avoid freezes, current implementations are still affected by this problem. For instance, the 2015 Conviva report shows that almost 25% of the analyzed HAS sessions are affected by at least one freeze [12].

Video quality Maximizing the video quality, given a certain available bandwidth, is beneficial for the user experience. Nevertheless, the same Conviva report highlights that low video quality still impacts 54% of the sessions. A clear trade-off can be identified between reducing freezes and maximizing quality, as requesting high-quality segments increases the chance of rebuffering events.

Quality switches A third objective to take into consideration is the amount of quality switches, as the adaptation heuristic can dynamically change the video quality to accommodate bandwidth variations. The impact of quality switches on user experience has been investigated by Hossfeld et al. [20]. The authors show that the time spent on the highest quality has the strongest impact on user experience, more than the number of switches itself. This result has also been confirmed in recent work by Tavakoli et al. [21]. This study also highlights that sudden reductions of video quality are perceived negatively by the user.

Latency In recent years, HAS has gained consistent momentum for the streaming of live events. In this scenario, end-to-end latency is of extreme importance to avoid the so-called *spoiler effect*. In fact, regular cable or satellite streams are usually characterized by a 5 to 10 seconds latency, which can instead grow to 30–60 seconds for HAS. This aspect entails that the viewing experience of HAS users can be spoiled by cable and satellite users, especially in case of important events. HAS solutions have been originally developed for Video-on-Demand (VoD), where latency is less of an issue, and are therefore suboptimal when it comes to live streaming. Together with live latency, also startup latency should be reduced as much as possible, especially when a user quickly switches between different video channels to search for some interesting content to watch.

Fairness Akshabi et al. are the first to report suboptimal behavior of HAS clients competing for shared bandwidth [13]. Fairness issues are not due to TCP dynamics, but mainly arise from the rate adaptation algorithms, as they decide on the actual rate to download. When multiple clients stream video at the same time, wrong bandwidth estimations can occur, due to the temporal overlap of the activity-inactivity periods of different clients. This wrong estimation subsequently affects the bitrate selection and thus the final QoE. Even though fairness is a system-wide characteristic rather than a user perceived QoE factor, it is often a desired property of the system, especially from the network and service provider point of view. Nevertheless, fairness issues can finally degrade user experience and should therefore be minimized.

Several attempts have been carried out to combine the aforementioned factors in one single combined QoE model for HAS [20–22]. Some of the proposed QoE models are also used in conjunction with network-based algorithms to improve

the delivery of HAS streams [23–28], which will be discussed in Section 2.3. De Vriendt et al. propose a model that is a linear combination of the average requested quality and its standard deviation, to keep into account quality switches [22]. The model is based on results obtained via a crowdsource experiment. A similar modelling approach is taken by Bentaleb et al. [24]. Besides the average quality and average quality difference between consecutive segments, the authors also include in their model a linear decreasing term depending on the number of freezes and the initial startup latency. Wang et al. propose an online QoE model, which is a function of the logarithm of the segment bitrate and the inverse of the freeze time [23]. The QoE model proposed by Mansy et al. depends instead on the screen resolution of the device, the viewing distance and the resolution of the quality played by the client [25]. Conceptually, the user-perceived quality degrades when the video resolution is lower than the screen resolution. Moreover, the structural similarity index is included in the model to account for the effect of the encoding bitrate. Essaili et al. employ a simple QoE model that is a linear function of the peak signal-to-noise ratio [28].

As a consensus on a specific QoE model has not been reached yet, we will use the aforementioned individual QoE factors when discussing existing works in Section 2.3, and to provide general guidelines on the best approach to use in Section 2.4.

2.3 Status of QoE-centric Management for HAS

In this section, we review the main works addressing the optimization of HAS services. We focus on three different research areas that go beyond purely client-based rate adaptation heuristics, namely: (i) server- and network-based solutions (Section 2.3.1), (ii) application level optimizations (Section 2.3.2) and (iii) TCP-layer modifications and emerging network architectures (Section 2.3.3).

2.3.1 Server- and Network-Based Optimizations

One of the success factors of the HAS principle has been its pull-based and decentralized nature, which allowed for an easy deployment. As reported in Section 2.2.2 though, this approach might not be sufficient to fully optimize users' QoE. For this reason, several approaches have been proposed, which use additional nodes located inside the network to help the delivery of the video [9, 30]. Following this trend, an extension of the MPEG-DASH standard has recently been published, called Server And Network assisted DASH (SAND) [29, 31]. In an MPEG-SAND architecture (Figure 2.3), DASH clients can communicate with DASH-Aware Network Elements (DANE). DANE nodes are located between the DASH clients and the server (which might be a DANE node itself), and are aware

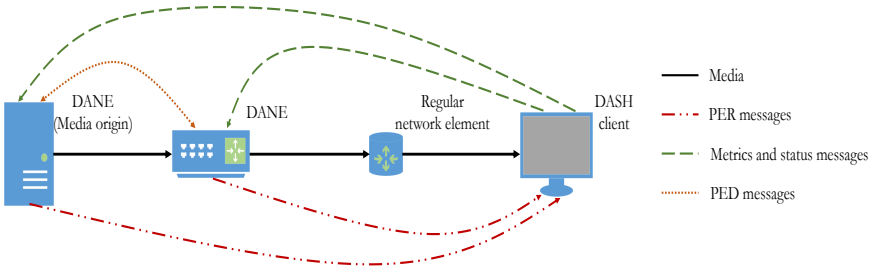


Figure 2.3: In an MPEG-SAND architecture, DASH-Aware Network Elements (DANE) can communicate among each other and with the clients to improve the end-to-end delivery of the video (adapted version of [29]).

of the underlying video streaming traffic. Based on the streaming and network conditions, these nodes can take decisions on the best way to handle DASH traffic in order to improve streaming performance (e.g., via bandwidth reservation, prioritization, rerouting, intelligent caching etc.). The MPEG-SAND standard defines the so-called SAND messages that can be exchanged between DANE elements and DASH clients. SAND status messages are used by the clients to report information to the network, so that the DANE elements are aware of the status of the clients. As an example, DASH clients can report QoE metrics that can be used by the network for monitoring purposes and/or to implement QoE-aware optimizations. Status messages also comprise information on the desired quality and bandwidth to facilitate resource sharing among competing clients, and hints on the future segments to request, to enable efficient caching strategies at the DANE nodes. Parameters Enhancing Reception (PER) messages are instead sent from the network to the clients in order to enhance and improve their quality adaptation. A DANE element can inform the clients about the available network throughput or communicate the segments that are already cached by the DANE. A client can therefore prefer to request these segments, as they are already available at the DANE and can be downloaded faster. Finally, DANE elements can communicate among each other using Parameters Enhancing Delivery (PED) messages. A PED message can, for example, be used by the server to communicate information about the streamed video to the network delivery elements. All SAND messages are delivered using the extensible markup language format over HTTP and follow a specific syntax defined by the standard [31].

Compared to client- or server-based solutions, network-assisted approaches are more difficult to deploy as they require: (i) a modification of the network nodes and (ii) an active collaboration between service and network providers. The MPEG-SAND standard represents an important enabler for these solutions, as it provides the set of messages the network, servers and clients can interchange to optimize the delivery of the video. Moreover, this collaboration can be beneficial for all

actors involved, in terms of generated revenues and reduced user churn, as pointed out from a techno-economic perspective by Ahmad et al. [32].

Being a fairly recent development of the MPEG-DASH standard, not all the works reported in this section follow the MPEG-SAND architecture. Nevertheless, they all share the same general principles and objectives defined by the standard, where a set of network nodes can communicate with each other and with the clients in order to optimize the end-to-end delivery of the video. In the following, we review the main works in this domain and categorize them based on the main approach used to improve the video delivery.

2.3.1.1 Traffic Rerouting

In approaches exploiting traffic rerouting, the performance of the paths connecting server and clients is continuously monitored. When certain conditions are detected (e.g., increased packet loss or congestion) a path recalculation occurs to reroute the video traffic and guarantee good streaming performance. Egilmez et al. have been among the first to investigate the use of rerouting for layered adaptive streaming [33]. A network controller, implemented using the Software-Defined Networking (SDN) principle, monitors the packet loss and delay of the network links. Each second, the optimal path for the video clients is computed, which minimizes a cost function based on packet loss and end-to-end delay. Two different optimization strategies are adopted for layered streaming. In the first solution, only the base layer is rerouted, as it is required at client-side to decode the video, while the enhancement layers are delivered over the shortest path. This strategy is sufficient to achieve higher quality compared to a shortest path delivery, when congestion is low. In the second solution, the enhancement layers are rerouted as well (with a lower priority compared to the base layer), which is effective when network congestion is high. Different service classes can also be introduced in the path computation optimization problem for layered video streaming [34]. In this work, network switches are equipped with one queue for each class, and the goal of the network controller is to maximize the service provider revenue, by rerouting traffic to satisfy as many high service class users as possible. While these rerouting strategies are executed periodically and for all the video flows in the network at the same time, Cetinkaya et al. recompute the optimal path per client, each time a new segment request is issued [35]. The server communicates to the network controller the bitrate of the requested segment. Based on this information and the links status, the controller reroutes the traffic to maximize the overall links throughput. This strategy is able to almost completely eliminate freezes compared to a best-effort delivery and to reach 15% higher bitrate. Despite that, scalability issues can emerge, as the controller has to operate on a per-client basis. To solve this problem, the path recomputation can also be triggered by the client itself [36]. In this case, when a client experiences a rebuffering event, an SDN controller col-

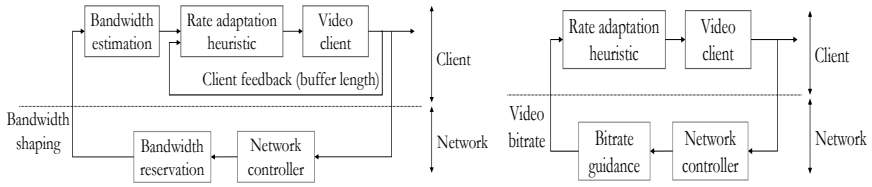
lects measurements on network links (in terms of bandwidth, packet loss and jitter) and runs a shortest path selection algorithm to find the best routing path for each flow. This approach can reduce the number of switches compared to a best-effort delivery and increase the time spent on the highest quality by 10%.

Traffic rerouting can also be indirectly achieved when multiple servers are available. In this scenario, the client can autonomously and dynamically select the best server to stream the video from [23, 37]. Bouten et al. use network characteristics to perform this selection [37]. During the start-up phase, the client polls the capabilities of the different servers by downloading a segment from each of these servers. The client then starts streaming from the server providing the best throughput. To avoid ending up in a local optimum, a probabilistic search is carried out by the client to test the performance of the available servers. This search is only executed when the buffer filling level is above a given threshold, to reduce the risk of freezes. The server selection strategy proposed by the authors is 12% worse than the optimal one in terms of QoE, which is modeled as proposed by De Vriendt et al. [22]. The user's QoE can be taken directly into account in the selection process [23]. The authors present an online QoE model, based on the bitrate of the segment and the freeze duration, which is used to evaluate the available servers. During the streaming session, the client evaluates the servers in terms of achieved QoE, and selects the one providing the best performance. Clients can also share QoE measurements among each other to enhance their selection process. Compared to a standard DASH client, the proposed approach results in 20% higher QoE.

Highlights for traffic rerouting: Rerouting allows to dynamically select the best path or server for the video delivery. As such, the video is always streamed over high-throughput, low-latency links, which increases the achieved video quality and reduces the amount of freezes. Relevant references: [33], [34], [35], [36], [37], [23].

2.3.1.2 Bandwidth Shaping and Bitrate Guidance

Bandwidth shaping and bitrate guidance techniques have attracted a lot of attention in the adaptive streaming community [9, 38]. Most of the works in this area share a similar architecture (Figure 2.4). A centralized node can collect information from both the network nodes and the clients. Network measurements include the available bandwidth and the number of streaming clients [9, 24, 25, 38–41]. Client-based information includes, for example, the screen dimension [25], the device type [38], the bitrates of the video [41] and buffer filling level [24]. The centralized node has therefore a comprehensive view of the streaming service, and can select the best bitrate for each client in order to maximize an objective function, which often models the users' QoE. The SDN principle is often used to



(a) Bandwidth shaping (b) Bitrate guidance

Figure 2.4: When bandwidth shaping is used (a), the network enforces a specific bandwidth for each client. In the bitrate guidance scenario (b), the network provides an explicit bitrate to the clients (adapted version of [9]).

provide a practical and scalable implementation of the developed solutions. Cofano et al. formally compare the performance of bandwidth shaping and bitrate guidance techniques [9]. A centralized node computes the optimal bitrates for the clients in order to obtain fairness in terms of video quality, estimated using the SSIM index. When bandwidth shaping is used, clients with similar optimal bitrates are assigned to a bandwidth slice. Despite this, the clients keeps the sole responsibility for the quality adaptation. In the bitrate guidance case, the optimal bitrates are explicitly communicated to the clients that download the corresponding segment. This approach provides the best overall results in terms of fairness and switching frequency compared to the bandwidth shaping approach. Similar conclusions are also drawn by Kleinrouweler et al. [41], who show that bitrate guidance is sufficient when streaming traffic is predominant. When DASH players have to compete with cross-traffic applications, the combination of slicing and guidance can provide the best results. Complex QoE models can be employed to improve the centralized bitrate selection [24, 25]. Bentaleb et al. use a QoE model depending on the average video quality, number of switches, freezes and startup latency [24]. The network controller selects the best video bitrate to maximize the QoE for each client. The optimal values are sent to the clients, which use it as an upper bound in their quality adaptation. The QoE model proposed by Mansy et al. depends instead on the screen resolution of the device, the viewing distance and the resolution of the quality played by the client [25]. Conceptually, the user-perceived quality degrades when the video resolution is lower than the screen resolution. Moreover, the SSIM is included in the model to account for the effect of the encoding bitrate. A measurement proxy located between the server and the client is investigated by Mok et al. [40]. The proxy estimates the highest available bitrate the clients can download based on the network conditions. This measurement is carried out using the media traffic itself, i.e., no explicit probing is needed. The clients use the quality level provided by the proxy as an upper bound. Moreover, a new adaptation heuristic is proposed to limit the number of switches. Similarly, Petrangeli et al. design a system of coordinated network proxies to help

HAS clients achieving fairness in a multiple-bottleneck network scenario [39]. The proxies compute an estimate of the fair bandwidth share of each client streaming video, and communicate this information to each client. The clients then request a quality that can both optimize their own QoE and the fairness of the whole system.

Highlights for bandwidth shaping and bitrate guidance: A centralized controller computes the optimal bitrates the clients should request, based on information collected from both the network nodes and the video clients. This decision is enforced by either creating a network slice or by informing the clients on the quality to request. These approaches allow to explicitly control the video quality the clients can achieve and improve the fairness of the system. Relevant references: [9], [38], [41], [24], [25], [40], [39].

2.3.1.3 Cross-Layer Optimization for Mobile Networks

When the last mile of the mobile network can be controlled, several optimizations can be carried out, by exploiting video-aware packet scheduler algorithms at the radio interface [42], channel and power allocation problems [43] or by monitoring the radio conditions [28]. In this case, the specific conditions and constraints of the mobile environment have to be taken into consideration when managing the radio resources. Chen et al. propose the AVIS system, which is composed of two elements [42]. An allocator centrally decides the bitrates to be requested by the clients, by maximizing an aggregate utility across all users while respecting the total number of resource transmitting blocks available at the base station. The utility is simply modeled as a logarithmic function of the requested bitrate and the number of quality switches. An enforcer schedules the transmission of the video flow packets so that the rate decided by the allocator is respected. When the AVIS systems is used, fairness among competing clients can be improved. Improving fairness is also one of the main objectives of Zhao et al. [43]. The authors define an optimization problem to allocate each user to a specific channel and transmission power level, in a layered video streaming scenario. More wireless resources are allocated to users whose segment playback deadline is close to expire or when the video quality contribution of requesting a specific layer is high. In other words, the marginal video quality contribution of higher layers decreases as the number of layers increases. Moreover, a DASH proxy stabilizes the quality decision process of the clients. The proxy computes the optimal quality level to request in order to maximize channel utilization and reduce switches. If the quality requested by the client is higher than the optimal one, the proxy overrides the client's decision by rewriting the HTTP GET request. A different approach for layered streaming is used by Fu et al. and Deng et al. [26, 27]. Video layers are assigned a priority based on the content characteristics. This priority depends on the visual quality gain of the layer and the bandwidth cost of transmitting the layer itself. The prior-

ity marking can happen at the server or in the network. The wireless base station can drop lower priority layers based on the channel condition of each user. This approach does not require modifying the radio interface scheduler and can provide 20% QoE improvements, with QoE modeled as a function of the number of received layers at the client. Essaili et al. use a QoE optimizer, located in the LTE network, which decides the transmission rate for each streaming client in order to maximize QoE, which is a function of the rate itself [28]. A penalty term is introduced to stabilize the rate and avoid QoE oscillations. Once the rate is decided, a proxy rewrites the HTTP GET request of the client to match the optimal rate. The easiest strategy is to choose the closest video representation for which the bitrate is below the optimal rate. The authors propose more complex solutions that take into account the buffer filling level of the clients. Compared to a classical HAS system, this approach results in up to 35% better QoE for the worst-case user.

Highlights for cross-layer optimization: Cross-layer optimizations take the condition of the radio interface into account to improve the delivery of HAS streams. Many of these solutions employ ad-hoc packet scheduling algorithms to improve the video quality delivered to the clients. As the optimization is carried out by a centralized node that is aware of the clients' conditions, fairness is also improved. Relevant references: [42], [43], [28], [26], [27].

2.3.1.4 Stream Prioritization

Prioritization has proven to be an effective way of optimizing QoE, especially in terms of video freezes [44]. Pu et al. propose a proxy node for mobile networks to concurrently optimize the delivery of multiple DASH streams [45]. DASH streams are assigned a different priority depending on the requested quality, which is considered a good indicator of the client status. Particularly, when the requested bitrate is below a specific threshold, which might indicate that the client is close to a freeze, the wireless scheduler assigns the stream a higher priority. The buffer status of the client can also be used to trigger prioritization [46]. In this work, a network controller prioritizes the packets belonging to video flows using a dedicated queue, when the client buffer drops below a certain threshold. The authors compare the performance of the priority queuing approach and a weighted fair queuing approach, and show that the latter performs better as it enables a more accurate bandwidth allocation to the flows. A similar approach is used by Petrangeli et al. [44]. A centralized SDN controller intercepts the HTTP GET requests of DASH clients and estimates their buffer status and requested quality using a machine learning algorithm. Based on this information and network measurements to detect congestion, video segments are enqueued in a prioritized queue to avoid future freezes. This approach can reduce freeze time with 45% when compared to classical HAS heuristics, without severely impacting the performance of cross-traffic applications.

Highlights for stream prioritization: A network controller is in charge of temporarily prioritizing the delivery of HAS streams, based on information as the requested quality or the clients' buffer level. These approaches are extremely effective in reducing video freezes. Relevant references: [44], [45], [46].

2.3.1.5 Content Delivery Network Orchestration and Caching Optimization

To satisfy the huge demand of video streaming requests, streaming providers have massively adopted Content Delivery Networks (CDN), where the video content is locally and temporally stored. By bringing the content closer to the end users, it is possible to reduce the load on the origin server and serve the video with lower latency and increased bandwidth. Casas et al. perform an experimental measurement study on the performance of YouTube's CDN infrastructure, with traffic collected in the core network of a national-wide mobile operator [47]. YouTube CDN servers are located at multiple autonomous systems, with some deployed directly inside the Internet Service Provider (ISP) networks. The measurement study shows that less than 10% of the flows served from CDN nodes inside the ISP network achieve a download throughput lower than 1 Mbps, while this value increases to 90% for servers located outside the ISP network. A large-scale measurement study is instead performed by Boettger et al. to model the CDN infrastructure of Netflix [48], called Open Connect. Particularly, the authors discover that half of Netflix's CDN nodes are deployed within ISPs, while the remaining nodes are deployed within Internet Exchange Points (IXP). CDN nodes are observed in 569 different ISP and 52 different IXP locations, respectively. Moreover, when looking at the deployment of the Open Connect platform over time, Netflix seems to mainly rely on IXP deployment as a first step, followed by a fine-grained ISP deployment. The authors argue that this strategy allows Netflix to reach a large user base during initial deployment, using IXP located servers. ISP located servers are instead used in a second phase to effectively reach a larger user base, which is often geographically scattered. A CDN orchestrator can be employed to manage such complex and geographically distributed infrastructures, as investigated by Mukerjee et al. [49]. The authors propose a control orchestration plane to optimize Conviva's C3 architecture, an Internet-scale CDN platform for the delivery of live videos [50]. A centralized controller, which operates on a timescale from tens of seconds to minutes, computes the optimal distribution trees for all the clients and videos in the different CDN clusters. The individual clusters apply a distributed algorithm on a sub-second timescale, to update the video forwarding strategy and keep into account local changes (e.g., link failures, workload changes). General caching strategies for video streaming can take into account the user and content characteristics (e.g., user mobility, content popularity) to better place the content in the CDN edge caches [51]. Krishnappa et al. exploit a particular user behavior to improve

the caching efficiency of YouTube videos [52]. Users are more inclined to watch videos on top of the related video list available on the YouTube web page. By rearranging this list to give preference to cached videos, cache hit rate is improved by a factor of 5. Caching strategies can also be improved when future user requests are known in advance [53]. For example, binge-watching has become typical on popular video streaming services, meaning that users tend to watch many episodes of the same TV show consecutively. This information can be used to estimate future video segment requests and improve the content placement in CDNs.

Unfortunately, caching can also negatively interfere with the particular dynamic of adaptive streaming clients, leading to frequent bitrate oscillations [54]. When a cache hit occurs for the requested segment, the throughput perceived by the client is generally high, which brings the client to request the next segment at a higher quality. If a cache miss occurs, the segment will have to be retrieved from the origin server. The increased latency in the segment delivery results in a lower bandwidth, which leads to a downshift in the requested quality. Transcoding can reduce this issue, by generating on-the-fly lower bitrates, starting from segments at higher bitrates located in the cache [55]. Transcoding techniques are examined in detail in Section 2.3.1.6. The problem of bitrate oscillations and caching is analyzed by Ge et al., in the context of edge computing in a 5G network [56]. A centralized node collects information on the requested segments and the conditions of the radio interface, and determines the most appropriate representations of individual segments to be cached. Only segments whose bitrate does not exceed the downlink capacity experienced by the mobile users are cached. Moreover, the replacement strategy tries to ensure segment continuity if a video is cached, in order to avoid cache misses. This approach can provide similar quality to the end users compared to a least frequently used cache strategy, but with 50% less switches. Li et al. analyze the problem of content placement in a wireless scenario, where the clients can stream the content from a set of edge servers, located close to the users [57]. The authors assume that the edge servers have to store all the segments belonging to a certain representation, if that particular representation needs to be stored. Each user in the system is associated with a distortion function, which decreases as the bitrate of the requested segment increases. The problem is therefore how to place the video representations in the edge servers in order to minimize the distortion experienced by the clients, assuming videos have different popularities and distortion functions, and the edge servers have limited cache capacity. Distortion is reduced by 20%, compared to a naive caching strategy that only caches most popular videos. A joint prefetching and caching architecture, called iPac, is proposed by Liang et al. [58]. A cache manager generates prefetching requests based on the segment requests from the clients. The bitrate of the prefetched segments is fixed to the one requested by the client. Moreover, the number of prefetched segments is limited to avoid wasting network resources if the requested bitrate would

change. A prefetching manager decides whether the prefetching request should be forwarded to the origin server or not. It is worth noting that not all the prefetching requests can be accepted, as they have to compete with the same bandwidth as standard cache miss requests, which have higher priority. This decision is based on how likely the prefetched segments will be requested in the near future. The authors propose an online prefetching algorithm that can perform at least half as good as an offline algorithm with complete knowledge of the system, in the worst case scenario. The iPac architecture can reach eight times higher byte-hit ratio compared to a least recently used cache strategy and increase user throughput by 50%.

Highlights for CDN orchestration and caching optimization: CDN orchestrator and intelligent content placement algorithms can be used to bring the most adapted video content (e.g., the most popular or the most personalized) closer to the end users. These solutions guarantee a higher throughput for streaming, which results in higher video quality and less freezes. Nevertheless, the succession of cache hit and cache miss complicates the bandwidth estimation process of the clients and can negatively affect the number of quality switches. Relevant references: [47], [48], [49], [51], [52], [53], [54], [55], [56], [57], [58].

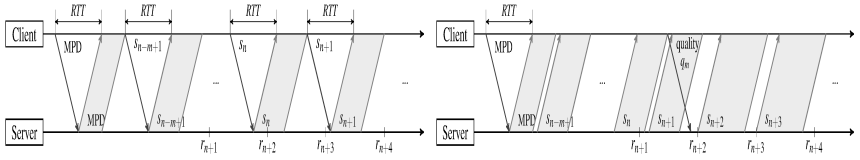
2.3.1.6 Server-Assisted Delivery

When the control on network components is limited, the server itself can assist the delivery of the video. Akhshabi et al. investigate a server-based traffic shaping algorithm, designed to reduce the quality fluctuations due to competing HAS clients sharing the same bottleneck [59]. The server identifies instable behavior of the clients (e.g., frequent quality increase/decrease) and limits the available download bandwidth to match the rate of the requested segment. This approach can eliminate the on-off pattern of the client requests, which is the main responsible for instability and unfairness in HAS. The server can also communicate directly with the client in order to efficiently schedule the client's HTTP GET requests to avoid rebuffering events [60]. The server foresees the occurrence of freezes by computing the download time of future segments over a specific time window. If a freeze is detected, a signal is sent to the client that consequently tries to ramp-up its buffer by downloading multiple segments in sequence. This approach reduces freezes up to 50% in wireless environments, compared to a client-based solution.

Many other works focus instead on the optimal choice of the encoding bitrates and online transcoding operations [61, 62]. In online transcoding, only few video representations are prepared by the server before the content is published; all the others are prepared on-the-fly when and if the clients request them. This approach allows to reduce the storage requirements in adaptive streaming, at the cost of an increased risk of rebuffering events, as the content has to be prepared online. Moreover, online transcoding is very computational intensive. Song et al.

develop a scheduling algorithm for power-efficient transcoding tasks [62]. An optimization problem is formulated to assign each transcoding job to the right CPU, whose frequency is adjusted to save energy while meeting the transcoding deadline. This approach can reduce power consumption up to 31% compared to the CPU scaling frequency algorithm used by Linux servers. To guarantee that the transcoding deadline is met when a client requests a segment, Ma et al. use a video transcoding time (VTT) estimation, based on a measurement study. When a video is uploaded to the server (e.g., in case of user-generated content), the encoding tasks are sorted based on the corresponding VTT and inserted into a low priority queue. Only when a segment is actually requested, it is moved into a high priority queue to guarantee a timely transcoding. Content popularity can also be taken into account [63]. In this work, all quality representations of videos whose popularity is higher than a rank threshold x_{high} are stored in edge servers. The highest quality only is stored for videos with popularity rank between x_{high} and x_{low}, in order to allow online transcoding. All the other videos are only available in the origin server. An optimization problem is formulated to minimize the storage and transcoding costs at the edge servers, and minimize bandwidth utilization between the origin and the edge servers when the content cannot be served by the latter. Using this approach, it is possible to reduce operational costs by 50%, when all the content is cached in advanced or is transcoded in the edge servers. Krishnappa et al. use a Markov model to predict which segments are going to be requested in the near future [64]. They also propose a hybrid transcoding policy, where only the first segment of the video is pre-encoded at all quality levels, while the remaining segments are transcoded online. The combination of this strategy and prediction allows to limit the rebuffering ratio to less than 1%. Moreover, transcoding operations are reduced by a factor of 10 compared to a solution where all the segments are pre-encoded. Joint transcoding and delivery strategies have also been proposed in [65]. A centralized node collects clients' preferences in terms of servers downstream bandwidth and estimates the number of requests for a particular segment, based on the segments requested in the near past. Based on this information, clients are redirected to more performing servers, while segments that are more likely to be requested are prioritized in the transcoding process. In a simulated environment, almost 45% of the clients are able to stream the video at a higher bitrate compared to a solution where the complete video is pre-encoded.

Highlights for server-assisted delivery: Server assistance is used to reduce client instability and therefore reduce the number of quality switches. Transcoding operations allow to reduce the storage requirements in CDN nodes, as the content is prepared on-the-fly if requested, but increase the risk of video freezes. Relevant references: [59], [60], [61], [62], [63], [64], [65].



(a) HTTP/1.1 (b) HTTP/2 server push

Figure 2.5: In classical HTTP/1.1 (a), each segment has to be retrieved independently and sequentially by the client. In HTTP/2 server push (b), the server can automatically push segments back-to-back. This behavior eliminates lost RTTs between segment requests and increases bandwidth utilization (adapted from [66]).

2.3.2 Application Level Optimizations

In case of over-the-top delivery, network elements are not accessible and cannot be used to support video clients. In this section, we review existing works that modify the application level of the HAS clients to improve the video delivery. We refer to the application level in a broad sense, both considering modifications of the actual application layer (e.g., HTTP/2 instead of HTTP/1.1) and enhancements of the client rate adaptation heuristic. Nevertheless, all the works in this section share two common aspects: (i) the main optimization is located at the client and (ii) they are compatible with any rate adaptation heuristic. In other words, these solutions can run on top of any HAS client implementation.

2.3.2.1 Adaptive Streaming over HTTP/2

The HTTP/2 protocol was originally developed to reduce the loading time of web pages, but it has also been applied to the delivery of multimedia content. Among the main features of HTTP/2, it is worth citing the native support of multiplexing (as opposed to HTTP/1.1), the possibility for the client to terminate the download of specific content and for the server to push content to the client, without an explicit request from the latter. Mueller et al. have been the first to analyze the impact of HTTP/2 in adaptive streaming [67]. Simply replacing HTTP/1.1 with HTTP/2 does not bring major improvements to the streaming performance and bandwidth utilization. Moreover, using HTTP/2 causes a slightly increased overhead compared to HTTP/1.1, as HTTP/2 is encrypted by default. By exploiting the specific functionalities of the HTTP/2 protocol, it is instead possible to consistently improve streaming performance. Huysegems et al. review the possible strategies that can be adopted in the context of adaptive streaming over HTTP/2 [68]. They propose ten different strategies based on stream termination, multiplexing and server push. Particularly, in server push, the server can proactively send (push) segments to a client based on previous requests (Figure 2.5). Especially for live streaming, using a push-based approach has multiple advantages. First, because subsequent

segments can be pushed back-to-back, lost RTT cycles between such segments are avoided, thus increasing the average link utilization and video quality. Second, a reduction of the live latency can be obtained, because video data is pushed as soon as it becomes available at the server. This is in contrast with traditional pull-based HAS approaches, where the client has to make an explicit request for new segments. In light of these advantages, the server push performance has been extensively investigated in order to reduce the live latency and the initial delay in HAS [10, 66, 69–72]. Wei et al. have been the first to investigate the performance of HTTP/2 server push in the context of adaptive streaming [69]. By reducing the segment duration from 5 to 1 second, the authors are able to reduce live latency by a factor 3. Despite this advantage, a trade-off is present between the number of pushed segments and streaming performance. Pushing many consecutive segments improves bandwidth utilization but reduces the capability of the clients to adapt to varying network conditions. To solve this issue, van der Hooft et al. propose to limit the number of pushed segments in flight to a certain value K [66]. This value is set based on the segment duration of the video and the network RTT. By combining this approach with segments with a sub-second duration, the startup and live delay are reduced by 30% compared to HTTP/1.1, with minimal impact on the freeze time. A similar approach is also proposed by Xiao et al. [10]. This study exploits a previous finding showing that pushing many consecutive segments can reduce the energy consumption up to 18% compared to a pull-based scenario [70]. Consequently, K is set to minimize the energy consumption on mobile devices, while the bitrate of the pushed segments is chosen to guarantee that the buffer does not drop below a given threshold. Moreover, stream termination is used to stop the push cycle in case a bandwidth drop is detected, and start a new one at a lower quality. This approach allows to eliminate video freezes and obtain 15% higher quality compared to classical adaptive streaming over HTTP/1.1. The buffer filling level can be included directly into the computation of the K value [71]. In this work, the optimal K is set in order to keep the buffer filling level above a specific threshold. To improve the bandwidth estimation during a push cycle, Cherif et al. use a WebSocket channel between server and clients, to communicate the beginning and the end of a push cycle. This approach is extremely beneficial when the content is retrieved from the local cache (as in a browser implementation), which can cause an overestimation of the available bandwidth.

Highlights for adaptive streaming over HTTP/2: The most exploited functionality of the HTTP/2 protocol is the server push mechanism, which results in higher link utilization compared to HTTP/1.1. This aspect results in a higher video quality and reduced live latency when server push is combined with sub-second segments. Nevertheless, these approaches might increase the number of freezes, as pushing segments reduces the bandwidth adaptation capability of the clients. Relevant references: [67], [68], [66], [70], [10], [71], [72].

2.3.2.2 Meta-Heuristics for Increased Client-Awareness

Streaming performance can be enhanced when the client can exploit additional information in the rate adaptation. This information, often referred to as context, can include positioning data (e.g., GPS) or historical information on the available bandwidth [73–75]. As an example, a mobile streaming client can be made aware of regions with limited or zero-bandwidth conditions, as a tunnel, to proactively avoid video freezes. In the event of future scarce connectivity, the client can ramp-up its buffer in order to compensate this condition. Liotou et al. propose an algorithm that deliberately degrades the quality requested by the client to ramp-up its buffer and provide a continuous playout in the low-connectivity zone [73]. When historical data on the available bandwidth are coupled with GPS position information, the bandwidth prediction of adaptive streaming clients is greatly improved, as shown by Riiser et al. [74]. The GPS and bandwidth information can be used to foresee future connectivity drops along a specific route, thus helping the actual quality adaptation. Particularly, the client uses both past perceived bandwidth samples and historic data to better plan the next quality to download. This approach results in a more stable quality behavior and fewer freezes. Optimizing the parameters of a given heuristic on-the-fly can also be beneficial [75]. In this work, a learning agent is designed to learn the best configuration of the heuristic parameters based on the bandwidth characteristics. As an example, the heuristic can be made more aggressive when bandwidth conditions are good and stable, or more conservative otherwise.

Highlights for meta-heuristics: Meta-heuristics exploit context information (as GPS and historical data on the available bandwidth) to improve the bitrate selection of the client. These works try to reduce the occurrence of freezes by using the context information to anticipate future bandwidth drops. Relevant references: [73], [74], [75].

2.3.2.3 Client-Based Prefetching

The performance of prefetching algorithms has been investigated in the scope of multi-view video streaming, where a user can dynamically switch between different video channels (e.g., different channels or different cameras capturing the same event). The goal in these scenarios is to minimize the channel switching delay [76, 77]. Prefetching strategies should carefully balance two contrasting objectives, namely maximizing quality and avoiding stalls in the played stream and prefetch new content from different channels to provide a seamless switch. Krishnamoorthi et al. exploit the typical on-off pattern of adaptive streaming clients for their prefetching algorithm [76]. Particularly, prefetching only occurs during off periods, to avoid interfering with the streaming of the played content. In the simplest solution, prefetching only occurs when the buffer filling level is above a

certain minimal threshold. This approach can be improved if the moment a certain channel is going to be played by the user is known in advance (e.g., by using a recommender system). In this case, prefetching can occur when the player is in an on period as well, to guarantee that the first segments of the new channel are already downloaded when the user would switch. This deadline-based strategy comes at the cost of a lower playback quality for the played video content and more freezes when bandwidth is scarce. To avoid this behavior, Carlsson et al. explicitly split the bandwidth between the played stream and the alternative streams [77]. Particularly, the currently played stream is allocated more bandwidth based on the buffer filling level, to avoid freezes. Given the bandwidth budget for prefetching, an optimization problem is proposed that maximizes the quality of the prefetched segments, assuming channels have different switching probabilities. Once the quality of the played and alternative streams is selected, the segment download is scheduled in a round-robin fashion. Conversely, prefetching can also be used in classical streaming scenarios to increase the video quality. This behavior has been experimentally observed by Sieber et al. for the browser-based YouTube player [78]. If a low quality level is requested but the adaptation heuristics detects an increase in the available bandwidth, one or more higher quality levels are prefetched in parallel, to present the user with a higher quality as soon as possible. Unfortunately, this approach results in almost 33% redundant traffic in the worst-case scenario.

Highlights for client-based prefetching: Prefetching algorithms download in the background video content that is likely going to be watched in the near future. This behavior minimizes latency in the context of channel-switching. Video quality and freezes can be negatively affected though, as prefetched content is in competion with the played stream. Relevant references: [76], [77], [78].

2.3.3 Transport Level Optimizations and Emerging Network Architectures

The TCP protocol was not originally designed to deliver video streaming applications, which are now responsible for a large portion of Internet traffic. As an example, Esteban et al. [79] show that HAS clients are often not capable of fully utilizing the available bandwidth of a link and that the on-off pattern of adaptive streaming can have a negative influence on the TCP congestion control. In this section, we review studies that optimize the transport layer to improve streaming performance and analyze works that propose and exploit new disruptive network infrastructures, as the Information Centric Networking (ICN) principle.

2.3.3.1 TCP Solutions for HAS

Enhanced TCP solutions for adaptive streaming generally follow two main trends. First, additional application-level information can be shared with the transport

layer to improve the scheduling of TCP packets, in order to primarily avoid freezes [80–82]. Second, Multi-path TCP (MPTCP) is employed to improve the aggregate throughput of the streaming client [83–85]. Lu et al. propose to improve the congestion detection mechanism of TCP [80]. In its standard implementation, TCP interprets all packet losses as an indication of network congestion. In wireless networks though, this behavior is sub-optimal, as losses can be due to wireless transmission errors rather than actual congestion. For this reason, the authors propose an algorithm that dynamically decides whether the TCP congestion control should be triggered or not. The algorithm exploits the inter-arrival time of TCP packets, which exhibits different characteristics based on the actual cause of the packet loss, namely congestion or wireless errors. Moreover, normal congestion control is triggered only if the lost packets contain an important frame of the video (i.e., I-frames or P-frames). The proposed approach results in 20% higher throughput than standard TCP. Incorporating the playout deadline of a video streaming packet can also improve the overall performance [82]. In this study, the playout deadline information is used to modify the congestion control parameters of TCP New Reno, in terms of increase and decrease of the congestion window. A flow characterized by an urgent deadline can temporarily adopt a more aggressive behavior, in order to achieve a higher throughput. Using this modification, the video clients can all meet their deadline, by using 15% less bandwidth than with normal TCP. The TCP modification proposed by the authors only takes place at the sender-side, which can facilitate its actual deployment. McQuistin et al. develop a new version of TCP, called Hollywood, that is wire-compatible with standard TCP in order to guarantee deployability [81]. Partial reliability is the main characteristic of this protocol: at the sender-side, TCP packets are retransmitted only if they can actually be used in the playout, i.e., if their playout deadline has not elapsed. If the deadline cannot be met, the payload of the retransmitted packets is substituted by another payload that can be used by the client. At the received side, TCP Hollywood removes head-of-line blocking, so that video segments can be passed to the application layer as soon as they are completed. Nazir et al. use a modified version of the TCP congestion control [86], called TCP Congestion Window Validation (CWV), to avoid that the congestion window of TCP is reduced too much during the off phase of an HAS client, where the client is not downloading any segment. This solution allows to obtain a higher and more stable TCP throughput, and consequently a better quality adaptation.

When multiple network interfaces are available, MPTCP can be used to increase the aggregate streaming throughput. Despite the logical expected advantages of this approach, MPTCP is not always beneficial for adaptive streaming [83]. The amount of bandwidth on the different interfaces and its variability can in fact have unexpected impacts on the streaming performance. James et al. show that only when the aggregate bandwidth is at least 40% higher than a specific bitrate, an

increase in quality can be obtained. Moreover, bandwidth variability on one interface can negatively influence the performance of the whole system. A cross-layer scheduler for multi-path TCP transmissions can reduce these disadvantages [85]. In this study, the cross-layer scheduler, which runs on top of the standard MPTCP packet scheduler, is aware of the structure and relationship among the different frames of the video and knows the buffer status of the client. Therefore, it can estimate the moment when a certain frame would need to be displayed. Moreover, the scheduler knows the conditions of the different interfaces, in terms of RTT, bandwidth and packet loss. More important frames can then be sent over the interface providing the best performance. By prioritizing the frames that are most likely to be received in time, the proposed solution can improve the average SSIM by 15% compared to the standard MPTCP scheduler. Work presented in [84] shows that user preferences should be taken into account when selecting the appropriate network interface for transmission. In this case, a cost function is associated with each interface, which depends on the specified preference. An optimization problem is defined, which minimizes the total delivery cost while respecting the playout deadlines of the video segments. For example, the cost function can depend on the energy consumption of a specific interface. When a Wi-Fi and an LTE interface are available, a higher cost can be associated to the latter. The authors show that their framework can reduce the amount of data transmitted over LTE by almost 40%, compared to standard MPTCP, without impacting the streaming performance. This result also has a positive impact on the energy consumption, which is reduced by 8%.

Highlights for TCP/MPTCP for HAS: TCP optimizations modify the congestion control and packet scheduling algorithms by incorporating video segments playout deadline information. This approach increases the achieved throughput and, therefore, the played video quality. A similar goal is obtained by MPTCP, which uses multiple network interfaces to stream the video. MPTCP solutions should be carefully designed, as the mutual influence of the different interfaces can negatively impact the amount of quality switches. Relevant references: [80], [82], [81], [83], [85], [84], [86].

2.3.3.2 ICN Approaches

Historically, the Internet has evolved in an ad-hoc manner where incremental patches were added to handle new requirements as they arose. This means that the underlying network model has not changed over the last decades, while the services using the Internet did so drastically. ICN is a disruptive network architecture that moves the traditional focus of a host-oriented communication model to a content-centric model, which can be extremely beneficial in adaptive streaming [87]. Particularly, ICN relies on location-independent naming schemes, in-network pervasive caching and content-based routing to allow an efficient distri-

bution of content over the network. Moreover, ICN nodes can seamlessly use all the available network interfaces to retrieve content, similarly to MPTCP. Content Centric Networking (CCN) and Named Data Networking (NDN) are typical instantiations of the ICN paradigm [88]. Nevertheless, ICN can also complicate the rate adaptation of HAS clients, as pointed out by Lederer et al. [87]. In ICN, the client is not aware of the node serving the content (e.g., the original server or one of the caches disseminated over the ICN network). This aspect complicates the estimation of the available bandwidth, similarly to when caches are used (Section 2.3.1.5). This issue has been experimentally confirmed by Liu et al. [89]. The authors also show that adaptive streaming over CCN results in 15% higher network overhead than regular DASH. Rainer et al. thoroughly investigate the interplay between different adaptation heuristics and interest forwarding strategies in NDN [11], where an interest represents the client request for a specific content. In this study, a theoretical framework is developed to find the upper bound for the average bitrate the clients can obtain, assuming the network and streaming characteristics are known a priori. Clients streaming over NDN can reach three times higher throughput than in classical TCP/IP networks, independently of the adopted heuristic or interest forwarding strategy. The best performance is reached when multiple interfaces are used to forward an interest and buffer-based heuristics are used at the client. These heuristics are in fact not susceptible to bandwidth miscalculations, which are likely in ICN due to multi-path transmissions and caching. To mitigate this issue, Ramakrishnan et al. investigate the possibilities of network coding in the context of multi-path interest forwarding [90]. In classical ICN, an interest forwarded over multiple interfaces could receive the same duplicated content over each interface. The authors exploit network coding to effectively aggregate the throughput available on multiple interfaces. Further improvements can be obtained when considering that each node has caching functionalities in ICN [91]. In this work, a video client can opportunistically retrieve video segments from both the server, using 3G/4G, and from other clients in a peer-to-peer fashion, using Wi-Fi. This solution results in better quality and reduced load on the mobile network. Intelligent caching strategies can also be developed in ICN [92, 93], with similar objectives as in regular DASH. Nevertheless, caching in ICN is advantaged by the naming structure of the interests. This aspect makes it easier for the network to understand what a user is watching and, most importantly, the relationship between different video segments [93]. Yu et al. use network condition information, available at the ICN nodes, and an estimate of future segment requests to prefetch content during off-peak congestion periods. Using this video- and network-aware prefetching strategy, the delivered quality increases by 20%, compared to a DASH system without prefetching.

Highlights for ICN approaches: ICN approaches combine the benefits of a pervasive fine-grained caching infrastructure and multi-path transmissions. These advantages entail a higher throughput compared to standard TCP/IP and, consequently, higher video quality and less freezes. As for caching though, ICN approaches can have a negative impact on the bandwidth estimation of the clients and, therefore, on the amount of quality switches. Relevant references: [87], [89], [11], [90], [91], [92], [93].

2.4 Recommendations

The goal of this section is to briefly summarize and discuss the different approaches presented in Section 2.3. Particularly, we provide some general guidelines regarding the benefits (or drawbacks) of each approach on several QoE factors, and the corresponding deployment complexity. A quick outlook of this analysis can be found in Table 2.1. For each QoE factor of interest, we indicate whether the analyzed approach has a positive or negative influence and the corresponding deployment complexity. It is worth noting that this classification only captures the main common trends among the works in the research areas presented in the previous section. In reality, each work is unique and presents advantages and disadvantages that cannot be completely generalized.

Traffic rerouting can consistently increase the obtained video quality and reduce freezes. To reroute traffic in the network, a centralized element has to know the status of all the network links involved in the delivery of the video, which allows to optimize the end-to-end conditions of the streaming path. This aspect represents a disadvantage of this solution as well, as it is often difficult to have such a global view on the network. Bandwidth shaping/bitrate guidance and cross-layer optimizations guidance techniques represent the ideal solutions when the quality delivered to each user has to be controlled in a fine-grained manner (e.g., to serve premium users or to guarantee fairness). These approaches are usually deployed on a bottleneck link located in the access or edge network, and are designed to select and enforce a specific quality for the streaming clients. This selection aims to optimize the revenue of the network operator, by maximizing the QoE of particular users, or provide fairness among competing clients. Bandwidth shaping techniques should be carefully designed as they could have a negative influence on quality switches [9, 41]. Cross-layer solutions can be challenging to deploy, as they require modifications of the radio interface. The applicability of bandwidth shaping techniques can instead benefit from the SDN principle and the MPEG-SAND standard. A similar consideration can be repeated for prioritization approaches, which are tailored on the specific task of reducing freezes. By bringing content closer to the users, smart caching strategies succeed in increasing the achieved throughput and consequently increase quality and avoid freezes. Nevertheless, the interference

Table 2.1: Overview of the different QoE-centric strategies. For each QoE factor, it is reported whether a particular approach has a positive (+), very positive (++) or negative (-) impact, or no impact (blank space). The deployment complexity of the solution is also reported (E: easy, M: medium: H: hard).

	QoE-centric strategy	*Quality*	*Switches*	*Freezes*	*Fairness*	*Latency*	*Complexity*	*References*
Sec 2.3.1	Rerouting	++		+			M/H	[33], [34], [35], [36], [37], [23]
	Bandwidth shaping	++	+/-		++		M	[9], [38], [41], [24], [25], [40], [39]
	Cross-layer optimization	++			++		M/H	[42], [43], [28], [26], [27]
	Prioritization			++			M	[44], [45], [46]
	CDN and caching	++	-	+			M	[47], [48], [49], [51], [52], [53], [54], [55], [56], [57], [58]
	Server-assistance		+	(-)			E/M	[59], [60], [61], [62], [63], [64], [65]
Sec 2.3.2	HTTP/2	+		-		++	E/M	[67], [68], [66], [70], [10], [71], [72]
	Meta-heuristics			+			E	[73], [74], [75]
	Prefetching	-		-		++	E	[76], [77], [78]
Sec 2.3.3	TCP/MPTCP	++	(-)	+			H	[80], [82], [81], [83], [85], [84], [86]
	ICN	++	-	++			H	[87], [89], [11], [90], [91], [92], [93]

between caching and HAS can have a negative impact on the switching behavior of the clients. This aspect should be carefully considered when designing caching strategies for HAS. Server-based solutions can effectively reduce switches when shaping techniques are used or help reducing storage costs when transcoding op-

erations are adopted. When transcoding is used, the risk of freezes increases as some video segments have to be generated online. Server- and network-assisted solutions are generally characterized by a medium to high deployment complexity, as they require modifications of network nodes. This aspect entails that the streaming provider and the network provider are willing to collaborate to optimize the behavior of HAS clients. Despite this, the presence of the MPEG-SAND standard and the benefits of such a collaboration for all actors involved [32] could be the drivers for an actual deployment of these solutions. Moreover, the MPEG consortium has started exploration activities related to network-distributed video coding and network-based media processing, which will rely on the MPEG-SAND specification [94].

Application level solutions are easier to deploy, as only the client has to be modified. In the case of HTTP/2, also the server has to be updated. Nevertheless, this aspect only marginally complicates an actual deployment, because servers and CDNs are starting providing HTTP/2 functionalities by default. Particularly, HTTP/2 push-based solutions can consistently reduce the live latency, increase bandwidth utilization and video quality, at the cost of reduced bandwidth adaptability that can result in more freezes. Prefetching is beneficial when the channel switching latency has to be minimized. A balance has to be found between prefetching many alternative channels and increasing the quality of the watched stream. Current works in the meta-heuristics domain mostly focus on reducing freezes by exploiting spatial and GPS information on network coverage.

Modifications at the transport layer focus on increasing the throughput achievable by streaming clients, by modifying the congestion control of standard TCP. These approaches also tend to be deadline-aware, i.e., they know when a particular video segment needs to be played at the client, to actively avoid a freeze. On a high level of abstraction, ICN-based solutions are characterized by pervasive in-network caching and multi-link usage. They therefore combine the advantages (and disadvantages) of caching solutions and MPTCP approaches. The optimizations presented in Section 2.3.3 are the most difficult to deploy nowadays, as they require an upgrade or modification of the current Internet infrastructure. Nevertheless, these optimizations can provide important insights for new transport protocols that are currently under development, as the Quick UDP Internet Connection (QUIC) protocol developed by Google [95].

2.5 Conclusions

Optimizing the QoE of HAS users is a complex and challenging task. Purely client-based rate adaptation heuristic might not reach optimal performance, as they only possess a local and decentralized knowledge of the streaming and network conditions. In this chapter, we therefore reviewed the status of existing works

in the adaptive streaming domain that go beyond classical adaptation heuristics. Particularly, we categorized these works in three groups, based on where the optimization takes place. First, server- and network-assisted solutions place additional intelligence in the network to support the delivery of the video. Traffic rerouting, bandwidth shaping techniques and caching represent the most typical examples in this space. This research area can greatly benefit from the new MPEG-SAND standard, which standardizes the possible messages network nodes can exchange to optimize HAS streams. Second, application level solutions can optimize the behavior of any adaptation heuristic by exploiting, for instance, the new features of the HTTP/2 protocol or prefetching techniques. Third, transport level approaches modify the congestion control algorithms or retransmission policy of TCP, to better support video traffic. All these different solutions have also been analyzed in terms of QoE benefits (e.g., quality, freezes, fairness etc.) and deployment complexity. In conclusion, a general *rule-them-all* solution for HAS, able to guarantee great and general benefits in terms of QoE factors and easy deployability, has not been developed yet. Network-based solutions provide the greatest advantages in terms of QoE improvements, as the network actively collaborates and supports the streaming clients. Unfortunately, this aspect also challenges the actual deployability of these approaches. Similar considerations can be repeated for transport level solutions. Application level optimizations are easy to deploy but can only provide consistent improvements on specific QoE factors, for example live latency. Nevertheless, the insights brought by this chapter can help future research efforts in the field of Internet video streaming.

Acknowledgment

Jeroen van der Hooft is funded by a grant of the Agency for Innovation by Science and Technology in Flanders (VLAIO). This research was performed partially within the imec ICON PRO-FLOW project (150223).

References

[1] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf, 2016.

[2] I. Sodagar. *The MPEG-DASH Standard for Multimedia Streaming Over the Internet*. IEEE MultiMedia, 18(4):62–67, April 2011.

[3] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials 99, 2014, 2014.

[4] J. Kua, G. Armitage, and P. Branch. *A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP*. IEEE Communications Surveys Tutorials, PP(99):1–1, 2017.

[5] Y. Sani, A. Mauthe, and C. Edwards. *Adaptive Bitrate Selection: A Survey*. IEEE Communications Surveys Tutorials, PP(99):1–1, 2017.

[6] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. *CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction*. In Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, pages 272–285, New York, NY, USA, 2016. ACM.

[7] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. *A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service*. In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 187–198, New York, NY, USA, 2014. ACM.

[8] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. *Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale*. IEEE Journal on Selected Areas in Communications, 32(4):719–733, April 2014.

[9] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. *Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming*. ACM Transactions on Multimedia Computing, Communications, and Applications, 2017.

[10] M. Xiao, V. Swaminathan, S. Wei, and S. Chen. *DASH2M: Exploring HTTP/2 for Internet Streaming to Mobile Devices*. In Proceedings of the 2016 ACM on Multimedia Conference, MM '16, pages 22–31, New York, NY, USA, 2016. ACM.

[11] B. Rainer, D. Posch, and H. Hellwagner. *Investigating the Performance of Pull-Based Dynamic Adaptive Streaming in NDN*. IEEE Journal on Selected Areas in Communications, 34(8):2130–2140, Aug 2016.

[12] CONVIVA. *Mid-Year 2015 Update: Conviva Viewer Experience Report*. http://www.conviva.com/conviva-viewer-experience-report/midyear-vxr-2015/, 2015.

[13] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptive Video Players over HTTP*. Image Commun., 27(4):271–287, April 2012.

[14] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. *Overview of the H.264/AVC video coding standard*. IEEE Transactions on Circuits and Systems for Video Technology, 13(7):560–576, July 2003.

[15] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the Scalable Video Coding Extension of the H.264/AVC Standard*. IEEE Transactions on Circuits and Systems for Video Technology, 17(9):1103–1120, Sept 2007.

[16] Appled HTTP Live Streaming. *Best Practices for Creating and Deploying HTTP Live Streaming Media for Apple Devices*. https://developer.apple.com/library/content/technotes/tn2224/_index.html.

[17] The Netflix Tech Blog. *Per-Title Encoding Optimization*. https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2.

[18] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. *BOLA: Near-optimal bitrate adaptation for online videos*. In IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pages 1–9, April 2016.

[19] C. Wang, A. Rizk, and M. Zink. *SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 1:1–1:12, New York, NY, USA, 2016. ACM.

[20] T. Hossfeld, M. Seufert, C. Sieber, and T. Zinner. *Assessing Effect Sizes of Influence Factors Towards a QoE Model for HTTP Adaptive Streaming*. In 2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX), pages 111–116, Sept 2014.

[21] S. Tavakoli, S. Egger, M. Seufert, R. Schatz, K. Brunnström, and N. García. *Perceptual Quality of HTTP Adaptive Streaming Strategies:*

Cross-Experimental Analysis of Multi-Laboratory and Crowdsourced Subjective Studies. IEEE Journal on Selected Areas in Communications, 34(8):2141–2153, Aug 2016.

[22] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for Estimating QoE of Video Delivered Using HTTP Adaptive Streaming*. In 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pages 1288–1293, May 2013.

[23] C. Wang, H. Kim, and R. Morla. *QoE Driven Server Selection for VoD in the Cloud*. In 2015 IEEE 8th International Conference on Cloud Computing, pages 917–924, June 2015.

[24] A. Bentaleb, A. C. Begen, and R. Zimmermann. *SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking*. In Proceedings of the 2016 ACM Conference on Multimedia Conference, MM 2016, Amsterdam, The Netherlands, October 15-19, 2016, pages 1296–1305, 2016.

[25] A. Mansy, M. Fayed, and M. Ammar. *Network-Layer Fairness for Adaptive Video Streams*. In 2015 IFIP Networking Conference (IFIP Networking), pages 1–9, May 2015.

[26] B. Fu, D. Staehle, G. Kunzmann, E. Steinbach, and W. Kellerer. *QoE-Based SVC Layer Dropping in LTE Networks Using Content-Aware Layer Priorities*. ACM Trans. Multimedia Comput. Commun. Appl., pages 7:1–7:23, August 2015.

[27] R. Deng and G. Liu. *QoE Driven Cross-Layer Scheme for DASH-Based Scalable Video Transmission over LTE*. Multimedia Tools and Applications, pages 1–25, 2017.

[28] A. E. Essaili, D. Schroeder, E. Steinbach, D. Staehle, and M. Shehada. *QoE-Based Traffic and Resource Management for Adaptive HTTP Video Delivery in LTE*. IEEE Transactions on Circuits and Systems for Video Technology, 25(6):988–1001, June 2015.

[29] E. Thomas, M. O. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey. *Enhancing MPEG DASH Performance via Server and Network Assistance*. SMPTE Motion Imaging Journal, 126(1):22–27, Jan 2017.

[30] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. *Network Assisted Content Distribution for Adaptive Bitrate Video Streaming*. In Proceedings of the 8th ACM on Multimedia Systems Conference, MMSys'17, pages 62–75, New York, NY, USA, 2017. ACM.

[31] *Information Technology – Dynamic Adaptive Streaming over HTTP (DASH) – Part 5: Server and Network Assisted DASH (SAND)*. Standard, International Organization for Standardization, 2017.

[32] A. Ahmad, A. Floris, and L. Atzori. *QoE-centric service delivery: A collaborative approach among OTTs and ISPs*. Computer Networks, 110:168 – 179, 2016.

[33] H. E. Egilmez, S. Civanlar, and A. M. Tekalp. *An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks*. IEEE Transactions on Multimedia, 15(3):710–715, April 2013.

[34] K. T. Bagci, K. E. Sahin, and A. M. Tekalp. *Queue-Allocation Optimization for Adaptive Video Streaming over Software Defined Networks with Multiple Service-Levels*. In 2016 IEEE International Conference on Image Processing (ICIP), pages 1519–1523, Sept 2016.

[35] C. Cetinkaya, E. Karayer, M. Sayit, and C. Hellge. *SDN for Segment Based Flow Routing of DASH*. In 2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin), pages 74–77, Sept 2014.

[36] H. Nam, K. H. Kim, J. Y. Kim, and H. Schulzrinne. *Towards QoE-Aware Video Streaming using SDN*. In 2014 IEEE Global Communications Conference, pages 1317–1322, Dec 2014.

[37] N. Bouten, M. Claeys, B. Van Poecke, S. Latré, and F. De Turck. *Dynamic Server Selection Strategy for Multi-Server HTTP Adaptive Streaming Services*. In 2016 12th International Conference on Network and Service Management (CNSM), pages 82–90, Oct 2016.

[38] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-Wide QoE Fairness Using Openflow-Assisted Adaptive Video Streaming*. In Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[39] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. *QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming*. ACM Trans. Multimedia Comput. Commun. Appl., 12(2):28:1–28:24, October 2015.

[40] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. *QDASH: A QoE-Aware DASH System*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 11–22. ACM, 2012.

[41] J. W. Kleinrouweler, S. Cabrero, and P. Cesar. *Delivering Stable High-Quality Video: An SDN Architecture with DASH Assisting Network Elements*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 4:1–4:10, New York, NY, USA, 2016. ACM.

[42] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang. *A Scheduling Framework for Adaptive Video Delivery over Cellular Networks*. In Proceedings of the 19th Annual International Conference on Mobile Computing and Networking, MobiCom '13, pages 389–400, New York, NY, USA, 2013. ACM.

[43] M. Zhao, X. Gong, J. Liang, W. Wang, X. Que, and S. Cheng. *QoE-Driven Cross-Layer Optimization for Wireless Dynamic Adaptive Streaming of Scalable Videos Over HTTP*. IEEE Transactions on Circuits and Systems for Video Technology, 25(3):451–465, March 2015.

[44] S. Petrangeli, T. Wu, T. Wauters, R. Huysegems, T. Bostoen, and F. D. Turck. *A machine learning-based framework for preventing video freezes in HTTP adaptive streaming*. Journal of Network and Computer Applications, 94(Supplement C):78 – 92, 2017.

[45] W. Pu, Z. Zou, and C. W. Chen. *Video Adaptation Proxy for Wireless Dynamic Adaptive Streaming over HTTP*. In 2012 19th International Packet Video Workshop (PV), pages 65–70, May 2012.

[46] T. Zinner, M. Jarschel, A. Blenk, F. Wamser, and W. Kellerer. *Dynamic Application-Aware Resource Management Using Software-Defined Networking: Implementation Prospects and Challenges*. In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–6, May 2014.

[47] P. Casas, P. Fiadino, A. Sackl, and A. D'Alconzo. *YouTube in the move: Understanding the performance of YouTube in cellular networks*. In 2014 IFIP Wireless Days (WD), pages 1–6, Nov 2014.

[48] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig. *Open Connect Everywhere: A Glimpse at the Internet Ecosystem through the Lens of the Netflix CDN*. CoRR, abs/1606.05519, 2016. Available from: http://arxiv.org/abs/1606.05519.

[49] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. *Practical, Real-time Centralized Control for CDN-based Live Video Delivery*. SIGCOMM Comput. Commun. Rev., 45(4):311–324, August 2015.

[50] A. Ganjam, J. Jiang, X. Liu, V. Sekar, F. Siddiqi, I. Stoica, J. Zhan, and H. Zhang. *C3: Internet-scale Control Plane for Video Quality Optimization*.

In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, pages 131–144, Berkeley, CA, USA, 2015. USENIX Association.

[51] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu. *Understanding Performance of Edge Content Caching for Mobile Video Streaming*. IEEE Journal on Selected Areas in Communications, 35(5):1076–1089, May 2017.

[52] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. *Cache-Centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches*. ACM Trans. Multimedia Comput. Commun. Appl., 11(4):48:1–48:20, June 2015.

[53] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latré, and F. De Turck. *Cooperative Announcement-Based Caching for Video-on-Demand Streaming*. IEEE Transactions on Network and Service Management, 13(2):308–321, June 2016.

[54] D. H. Lee, C. Dovrolis, and A. C. Begen. *Caching in HTTP Adaptive Streaming: Friend or Foe?* In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 31:31–31:36, New York, NY, USA, 2014. ACM.

[55] H. Ahlehagh and S. Dey. *Adaptive Bit Rate Capable Video Caching and Scheduling*. In 2013 IEEE Wireless Communications and Networking Conference (WCNC), pages 1357–1362, April 2013.

[56] C. Ge, N. Wang, S. Skillman, G. Foster, and Y. Cao. *QoE-Driven DASH Video Caching and Adaptation at 5G Mobile Edge*. In Proceedings of the 3rd ACM Conference on Information-Centric Networking, ACM-ICN '16, pages 237–242, New York, NY, USA, 2016. ACM.

[57] C. Li, P. Frossard, H. Xiong, and J. Zou. *Distributed Wireless Video Caching Placement for Dynamic Adaptive Streaming*. In Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '16, pages 2:1–2:6, New York, NY, USA, 2016. ACM.

[58] K. Liang, J. Hao, R. Zimmermann, and D. K. Y. Yau. *Integrated Prefetching and Caching for Adaptive Video Streaming over HTTP: An Online Approach*. In Proceedings of the 6th ACM Multimedia Systems Conference, MMSys '15, pages 142–152, New York, NY, USA, 2015. ACM.

[59] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. *Server-Based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players*. In

Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '13, pages 19–24, New York, NY, USA, 2013. ACM.

[60] S. Colonnese, F. Cuomo, R. Guida, and T. Melodia. *Performance Evaluation of Sender-Assisted HTTP-Based Video Streaming in Wireless Ad Hoc Networks*. Ad Hoc Networks, 24, Part B:74 – 84, 2015.

[61] L. Toni, R. Aparicio-Pardo, K. Pires, G. Simon, A. Blanc, and P. Frossard. *Optimal Selection of Adaptive Streaming Representations*. ACM Trans. Multimedia Comput. Commun. Appl., 11(2s):43:1–43:26, February 2015.

[62] M. Song, Y. Lee, and J. Park. *Scheduling a Video Transcoding Server to Save Energy*. ACM Trans. Multimedia Comput. Commun. Appl., 11(2s):45:1–45:23, February 2015.

[63] Y. Jin, Y. Wen, and C. Westphal. *Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud: an Analytical Approach*. IEEE Transactions on Circuits and Systems for Video Technology, 25(12):1914–1925, Dec 2015.

[64] D. K. Krishnappa, M. Zink, and R. K. Sitaraman. *Optimizing the Video Transcoding Workflow in Content Delivery Networks*. In Proceedings of the 6th ACM Multimedia Systems Conference, MMSys '15, pages 37–48. ACM, 2015.

[65] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang. *Joint Online Transcoding and Geo-Distributed Delivery for Dynamic Adaptive Streaming*. In IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, pages 91–99, April 2014.

[66] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Journal of Network and Systems Management, pages 1–28, 2017.

[67] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner. *Dynamic Adaptive Streaming over HTTP/2.0*. In 2013 IEEE International Conference on Multimedia and Expo (ICME), pages 1–6, July 2013.

[68] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck. *HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming*. In Proceedings of the 23rd ACM International Conference on Multimedia, MM '15, pages 541–550, New York, NY, USA, 2015. ACM.

[69] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 37:37–37:42, New York, NY, USA, 2014. ACM.

[70] S. Wei, V. Swaminathan, and M. Xiao. *Power Efficient Mobile Video Streaming using HTTP/2 Server Push*. In 2015 IEEE 17th International Workshop on Multimedia Signal Processing (MMSP), pages 1–6, Oct 2015.

[71] D. V. Nguyen, H. T. Le, P. N. Nam, A. T. Pham, and T. C. Thang. *Request Adaptation for Adaptive Streaming over HTTP/2*. In 2016 IEEE International Conference on Consumer Electronics (ICCE), pages 189–191, Jan 2016.

[72] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. *DASH Fast Start Using HTTP/2*. In Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '15, pages 25–30, New York, NY, USA, 2015. ACM.

[73] E. Liotou, T. Hossfeld, C. Moldovan, F. Metzger, D. Tsolkas, and N. Passas. *Enriching HTTP Adaptive Streaming with Context Awareness: A Tunnel Case Study*. In 2016 IEEE International Conference on Communications (ICC), pages 1–6, May 2016.

[74] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning*. ACM Trans. Multimedia Comput. Commun. Appl., 8(3):24:1–24:19, August 2012.

[75] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck. *A Learning-Based Algorithm for Improved bandwidth-Awareness of Adaptive Streaming Clients*. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 131–138, May 2015.

[76] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. *Bandwidth-Aware Prefetching for Proactive Multi-Video Preloading and Improved HAS Performance*. In Proceedings of the 23rd ACM International Conference on Multimedia, MM '15, pages 551–560, New York, NY, USA, 2015. ACM.

[77] N. Carlsson, D. Eager, V. Krishnamoorthi, and T. Polishchuk. *Optimized Adaptive Streaming of Multi-Video Stream Bundles*. IEEE Transactions on Multimedia, PP(99):1–1, 2017.

[78] C. Sieber, A. Blenk, M. Hinteregger, and W. Kellerer. *The cost of aggressive HTTP adaptive streaming: Quantifying YouTube's redundant traffic*. In

2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1261–1267, May 2015.

[79] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. *Interactions Between HTTP Adaptive Streaming and TCP*. In Proceedings of the 22Nd International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV '12, pages 21–26, New York, NY, USA, 2012. ACM.

[80] Z. Lu, V. S. Somayazulu, and H. Moustafa. *Context-Adaptive Cross-layer TCP Optimization for Internet Video Streaming*. In 2014 IEEE International Conference on Communications (ICC), pages 1723–1728, June 2014.

[81] S. McQuistin, C. Perkins, and M. Fayed. *TCP Goes to Hollywood*. In Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '16, pages 5:1–5:6, New York, NY, USA, 2016. ACM.

[82] M. Claeys, N. Bouten, D. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, S. Latré, and F. De Turck. *Deadline-Aware TCP Congestion Control for Video Streaming Services*. In 2016 12th International Conference on Network and Service Management (CNSM), pages 100–108, Oct 2016.

[83] C. James, E. Halepovic, M. Wang, R. Jana, and N. K. Shankaranarayanan. *Is Multipath TCP (MPTCP) Beneficial for Video Streaming over DASH?* In 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 331–336, Sept 2016.

[84] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. *MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath*. In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, CoNEXT '16, pages 129–143, New York, NY, USA, 2016. ACM.

[85] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. *Cross-layer Scheduler for Video Streaming over MPTCP*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 7:1–7:12, New York, NY, USA, 2016. ACM.

[86] S. Nazir, Z. Hossain, R. Secchi, M. Broadbent, A. Petlund, and G. Fairhurst. *Performance Evaluation of Congestion Window Validation for DASH Transport*. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 67:67–67:72. ACM, 2014.

[87] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner. *An Experimental Analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks*. In 2013 IEEE International Conference on Multimedia and Expo (ICME), pages 1–6, July 2013.

[88] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. *A survey of information-centric networking*. IEEE Communications Magazine, 50(7):26–36, July 2012.

[89] Y. Liu, J. Geurts, J. C. Point, S. Lederer, B. Rainer, C. Mueller, C. Timmerer, and H. Hellwagner. *Dynamic Adaptive Streaming over CCN: A Caching and Overhead Analysis*. In 2013 IEEE International Conference on Communications (ICC), pages 3629–3633, June 2013.

[90] A. Ramakrishnan, C. Westphal, and J. Saltarin. *Adaptive Video Streaming over CCN with Network Coding for Seamless Mobility*. In 2016 IEEE International Symposium on Multimedia (ISM), pages 238–242, Dec 2016.

[91] B. Han, X. Wang, N. Choi, T. Kwon, and Y. Choi. *AMVS-NDN: Adaptive Mobile Video Streaming and Sharing in Wireless Named Data Networking*. In 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 375–380, April 2013.

[92] W. Li, S. Oteafy, and H. Hassanein. *Rate-Selective Caching for Adaptive Streaming over Information-centric Networks*. IEEE Transactions on Computers, PP(99):1–1, 2017.

[93] Y.-T. Yu, F. Bronzino, R. Fan, C. Westphal, and M. Gerla. *Congestion-Aware Edge Caching for Adaptive Video Streaming in Information-Centric Networks*. In 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pages 588–596, Jan 2015.

[94] K. Park, I. Bouazizi, and Y. Xu. *Use cases and draft requirements for Network Based Media Processing*. https://mpeg.chiariglione.org/standards/exploration/network-based-media-processing/requirements-network-based-media-processing-v1, 2017.

[95] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kuehlewind. *Innovating Transport with QUIC: Design Approaches and Research Challenges*. IEEE Internet Computing, 21(2):72–76, Mar 2017.

3

QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming

S. Petrangeli, J. Famaey, M. Claeys, S. Latré and F. De Turck.

Published in ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), November 2015.

⋆ ⋆ ⋆

This chapter tackles one of the problems still affecting HTTP Adaptive Streaming (HAS) systems, namely fairness. Concretely, this means that different HAS clients negatively influence each other as they compete for shared network resources. To solve this problem, a novel rate adaptation algorithm is proposed, capable of increasing clients' Quality of Experience (QoE) and achieving fairness in a multi-client setting. A key element of this approach is an in-network system of coordination proxies in charge of facilitating fair resource sharing among clients. The strength of this approach is threefold. First, fairness is achieved without explicit communication among clients and thus no significant overhead is introduced into the network. Second, the system of coordination proxies is transparent to the clients, i.e., the clients do not need to be aware of its presence. Third, the HAS principle is maintained, as the in-network components only provide the clients with new information and suggestions, while the rate adaptation decision

remains the sole responsibility of the clients themselves. This approach is evaluated through simulations, under highly variable bandwidth conditions and in several multi-client scenarios. The proposed approach can improve fairness up to 80% compared to state-of-the-art HAS heuristics in a scenario with three networks, each containing 30 clients streaming video at the same time.

3.1 Introduction

Nowadays, video streaming applications are responsible for the largest portion of the traffic exchanged over the Internet. Particularly, HTTP Adaptive Streaming (HAS) protocols have become very popular due to their flexibility, and can therefore be considered as the de facto standard for video streaming services. Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming and MPEG Dynamic Adaptive Streaming over HTTP (DASH) are examples of available HAS technologies. In an HAS architecture, video content is stored on a server as segments of fixed duration at different quality levels. Each client can request the segment at the most appropriate quality level on the basis of the local perceived bandwidth. In this way, video playback dynamically changes according to the available resources, resulting in a smoother video streaming experience. The main disadvantage of current HAS solutions is that the heuristics used by the clients to select the appropriate quality level underperform in a multi-client scenario [1–3]. In a real scenario, multiple clients simultaneously request content from the HAS server. Often, clients have to share the same network resources (e.g., a link) and issues concerning fairness among them appear, meaning that the presence of a client has a negative impact on the performance of others. As reported by Akhshabi et al. [1], fairness issues are not due to TCP dynamics, but mainly arise from the rate adaptation algorithms, as they decide on the actual rate to download. When multiple clients stream a video at the same time, a wrong bandwidth estimation can occur, due to the temporal overlap of the activity-inactivity periods of different clients. Indeed, in steady-state conditions, HAS clients usually leave a time gap between the download of a segment and the request of a new one, to avoid filling up their video buffer to a too large level. This wrong estimation subsequently affects the bitrate selection and thus the clients' Quality of Experience (QoE). This problem is aggravated by the uncoordinated nature of current HAS heuristics. This entails they are not aware of the presence of other clients nor can they adapt their behavior to deal with it.

In this chapter, we investigate the aforementioned problems arising in a multi-client setting. Particularly, we present a fair HAS client able to achieve smooth video playback, while coordinating with other clients in order to improve the fairness of the entire system. This goal is reached with the aid of an in-network-based system of *coordination proxies*, in charge of collecting measurements on the net-

work conditions. This information is then used by the clients to refine their quality decision process and develop a fair behavior.

The main contributions of this chapter are threefold. First, we present a new HAS heuristic called FINEAS (Fair In-Network Enhanced Adaptive Streaming) able to select the best quality depending on the network conditions, in order to provide a smooth video streaming and improve fairness. Particularly, our heuristic is able to increase the average requested quality level compared to current HAS heuristics and avoid video freezes, while guaranteeing similar QoE to all the clients streaming video, i.e., fairness. Second, we design an in-network-based system to help clients coordinate their behavior, which does not require explicit client-to-client communication or a centralized decision process. Consequently, the quality level selection can still be performed locally and independently by each client, without any modification to the general HAS principle. Third, detailed simulation results are presented to characterize the gain of the proposed framework compared to state-of-the-art HAS heuristics.

The remainder of this chapter is structured as follows. Section 3.2 reports related work on HAS and multi-client algorithms. Section 3.3 details the proposed multi-client HAS framework, both from an architectural and algorithmic point of view. In Section 3.4, we evaluate our solution through simulation and show its effectiveness compared to current HAS heuristics. Section 3.5 presents the main conclusions. The online appendix presents an analytical formulation of the general fairness problem.

3.2 Related Work

Akhshabi et al. present an analysis of the performance and drawbacks of some commercially available HAS heuristics, such as Microsoft Smooth Streaming, Netflix and Adobe players [4]. It is shown that current heuristics perform quality selection sub-optimally. Particularly, they fail to adapt to rapid bandwidth changes, leading to drops in the client play-out buffer and unnecessary quality switches. They also analyze the performance of two competing HAS clients sharing the same bottleneck, and point out that they are not able to develop a fair behavior. Similar considerations are reported by Mueller et al. when testing different HAS implementations using real bandwidth traces collected on a mobile network [5]. They also point out that the Microsoft Smooth Streaming client is able to achieve the highest average bitrate as well as a low number of quality switches.

Many HAS rate adaptation heuristics have been proposed to alleviate the problems highlighted in the previous paragraph. Zhuo et al. present a control theory-based HAS client where the buffer filling level of the client is controlled [6]. A similar approach is also studied by Tian et al. [7]. Adzic et al. propose to add additional information into the video segments to enhance the quality decision

algorithm [8]. The client can then decide to switch up or down depending on the effect on bitrate and the intrinsic quality of the next segment to download. Jarnikov et al. study an algorithm based on a Markov Decision Process (MDP), which requires offline training [9]. Xiang et al. use a similar approach for Scalable Video Coding streaming and adopt a Markov model to describe the variations of wireless channel conditions [10]. Also the work presented by Claeys et al. is based on an MDP and Reinforcement Learning theory [11]. In this case, the solution to the MDP is computed online by means of the Q-Learning algorithm, without any a priori knowledge. All these algorithms share a common drawback. Although effective in a single-client scenario, they are not explicitly designed to deal with the presence of multiple clients streaming video simultaneously over a shared network medium. This means that they can fail in these circumstances, as shown by Akhshabi et al. [1]. They report that unfairness is mainly caused by the quality adaptation algorithms themselves, since they are not designed to explicitly cope with a multi-client scenario. It is shown that a relevant cause of unfairness is the temporal overlap of the activity-inactivity periods of different clients, since this can lead to wrong bandwidth estimations.

Based on this consideration, most of the algorithms designed to improve fairness in HAS focus on the modification of the time interval at which clients request a new segment. Villa et al. implement a modified version of the Microsoft ISS Smooth Streaming (MSS) client, randomizing the quality selection decision interval [12]. The results show that this technique can improve fairness, but the randomization characteristics have to be selected carefully. A similar approach is also used by Jiang et al. [13]. In this case, the next segment download is randomized taking into account the buffer status, in order to avoid freezes. Moreover, a stateful bitrate selection is introduced to allow clients with a low quality to increase it more aggressively. The authors theoretically prove that this selection allows convergence of the clients' bitrates. In the work by Li et al., the download of a chunk is scheduled to obtain a continuous average data rate sent over the network and to maintain the buffer level close to a given threshold [2]. The segment fetch time is used by Liu et al. to decide which quality level to request [14]. They try to improve fairness by allowing recently joined clients to behave more aggressively. De Cicco et al. propose the ELASTIC client, which avoids activity-inactivity periods by downloading segments in order to maintain the buffer level close to a set-point [15]. The ELASTIC client is mainly designed to obtain fairness from the network point of view but not to maximize QoE, since quality switches are not included in the rate adaptation heuristic. This results in a high number of switches, even in a fixed bandwidth scenario, and consequently low QoE. The main problem of the presented algorithms is that they are purely client-based, i.e., no coordination is envisaged among the clients. Although this aspect simplifies the design and implementation of the algorithms, it does not allow solving the fairness problem

completely. The lack of coordination mechanisms entails that the fairness problem has to be solved at design time and that the solution has to be embedded in the client's heuristic. Since fairness is inherently an online problem, which depends on the number of clients, network topology and bandwidth conditions, the obtained solution is sub-optimal. As an example, Mueller et al. pointed out that the network infrastructure, as a system of caching proxies, has a negative impact on the bandwidth estimation process of the clients, and thus also on their performance in terms of QoE and fairness [16]. This type of information, which is not known at design time and thus not available to purely client-based algorithms, affects the effectiveness of the aforementioned solutions.

A first approach to tackle this problem would be a centralized solution, where the video server decides on the quality level to return. This solution is investigated by De Cicco et al. and Kuschnig et al. [17, 18]. De Cicco et al. propose a control theory-based algorithm to control the length of the sender buffer, placed at the server, in order to select the most appropriate quality level [17]. In the work presented by Kushnig et al., the server estimates the bandwidth of the TCP connection considering the number of bytes transmitted in a time unit [18]. This estimation is performed on a per-GOP (Group Of Pictures) basis. Even if a centralized approach would facilitate the computation of an optimal global fair policy, it is not scalable if the number of clients to serve increases. Moreover, a centralized approach alters the classical HAS principle.

In order to solve the scalability issue, we adopt in this chapter an in-network approach. In in-network algorithms, intermediary nodes are placed in the network to collect information regarding the available bandwidth and influence the behavior of HAS clients. An example of in-network adaptation is given by Bouten et al. [19]. A system of network proxies periodically solves an optimization problem to determine the maximum quality the clients can download, based on the current network status and an objective function. Houdaille et al. introduce a bandwidth manager inside the home gateway to manage the flows belonging to the different clients [20]. Based on clients' characteristics and network conditions, the bandwidth manager determines the target bitrate for each client, in order to fairly share the available bandwidth. The bandwidth manager then applies traffic shaping techniques to limit the bandwidth of each client and drive it to request the target bitrate. A similar approach is investigated by Georgopoulos et al. [21]. A centralized OpenFlow controller is used to allocate the bandwidth for each streaming device, in order to obtain fairness from a QoE point of view. The authors present a model to correlate video bitrate with video quality, which is used by the controller in the bandwidth allocation process. Also Ma et al. implemented a bandwidth manager to determine the quality levels to assign to the clients, based on the available bandwidth and their subscription level [22]. Mok et al. introduce a measurement proxy between the clients and the video streaming server, which

estimates the available bandwidth and decides the highest possible quality level the clients can support [23]. Based on this information and their buffer status, the clients decide the most appropriate quality level to request. Essaili et al. propose a QoE optimizer for wireless networks that computes the optimal rate for the streaming clients, based on the wireless channel conditions [24]. This value is then used by a QoE proxy, in charge of intercepting and rewriting clients' requests to match the requested quality level with the optimal rate. The principal disadvantage of the aforementioned algorithms is the active role of the in-network elements in the quality decision process. This aspect entails an alteration of the classical HAS principle, as the network *de-facto* decides which quality level the clients can download. Even if different clients will select quality differently, at steady-state all clients will converge to the quality level enforced by the network. Moreover, these solutions present robustness issues in case of fault or malfunctioning of the in-network elements.

Based on these considerations, we develop a new client-based rate adaptation heuristic to optimize the QoE delivered to the clients, together with an in-network coordination mechanism to improve fairness, which can operate in face of multiple bottleneck links. This approach presents two advantages. First, the in-network computation can be kept very simple and consequently not computationally demanding, since the quality decision algorithm still runs at the client. Second, it is more robust in case of fault or malfunctioning of the network equipment, as the client can continue to operate (at a sub-optimal level) without the in-network system. The concept of an in-network system to help clients obtaining fairness has been firstly introduced in our previous work [3]. The difference with respect to the work presented in this chapter is twofold. First, the in-network system presented in our previous work is composed of a single centralized node, which collects and aggregates statistics on clients' QoE performance. Even though the presented approach showed promising results, further experiments have shown unsatisfactory performance in complex scenarios, mainly due to: (i) the centralized nature of the solution, which is not able to handle the presence of multiple bottlenecks and (ii) the aggregation of QoE statistics, which is not trivial if clients are located in different sub-networks. In this chapter instead, we propose a distributed in-network system, able to cope with the presence of multiple bottlenecks, computing an estimate of the clients' fair bandwidth share on bottleneck links. This information is provided to the clients, which use it in their heuristics to achieve fairness. Second, we design a completely new rate adaptation heuristic, called FINEAS, to maximize client's QoE and exploit the information provided by the in-network system to achieve fairness.

3.3 Proposed fair HAS framework

The problem we propose to investigate in this chapter is twofold. First, clients have to obtain the highest possible video quality. Second, they have to show similar per-

formance if they share bottleneck links, i.e., fairness. Based on this consideration, all the clients sharing the same bottlenecks should act fairly, even if they belong to different networks. An analytical formulation of the general fairness problem is presented in the appendix. In order to maximize the QoE delivered to the clients and achieve fairness, we present the FINEAS (Fair In-Network Enhanced Adaptive Streaming) heuristic. The FINEAS heuristic runs at the clients and performs the quality level selection based on three inputs: the local perceived bandwidth, the video player buffer status and the so-called *fairness signal*. The fairness signal is an additional measure introduced to achieve fairness, obtained when the client downloads a segment. The fairness signal is computed by a system of network nodes, called *coordination proxies*, and represents an estimate of the fair bandwidth share of all the clients streaming video. In the remainder of this section we provide an architectural overview of the proposed framework (Section 3.3.1) and detail the implementation of the FINEAS heuristic (Section 3.3.2) and of the fairness signal computation (Section 3.3.3).

3.3.1 Architectural Overview

As introduced previously, the system of coordination proxies is in charge of helping clients achieving fairness, by computing an estimate of the fair bandwidth share for all the clients streaming video, even if they belong to different networks. In order to maintain scalability, the computation of the fairness signal is performed periodically and in a hierarchical way by the coordination proxies. A generic coordination proxy P receives an estimate of the fairness signal from its parent node and computes a new estimate of the fairness signal for each of its child proxies. This estimate is computed by monitoring the available bandwidth for HAS traffic on the links connecting proxy P to its child nodes. A root proxy triggers the computation of the fairness signal. At initialization, the coordination proxies start monitoring the bandwidth for HAS traffic. After T_{fair} seconds, given T_{fair} is the fairness signal computation period, the root proxy forwards its first estimate of the signal and triggers its computation for the entire system. In order to limit overhead, the calculated fairness signal can be added as an HTTP header field and returned to the clients when delivering the next segment to play. Particularly, the clients translate the fairness signal into a *reference* quality level, representing the theoretical quality level to request in order to obtain perfect fairness among the clients. This reference gives an indication on the best quality level to achieve fairness, rather than determining the actual quality to be requested. The reason for this behavior is twofold. First, directly requesting the reference quality level would be optimal from the fairness point of view but not from the QoE point of view, because of the frequent switches that would occur. Second, directly requesting the reference quality level would alter the classical HAS principle, as the decision on the quality level to download would no longer be carried out by the clients.

The main advantage of this hybrid approach is threefold. First, no commu-

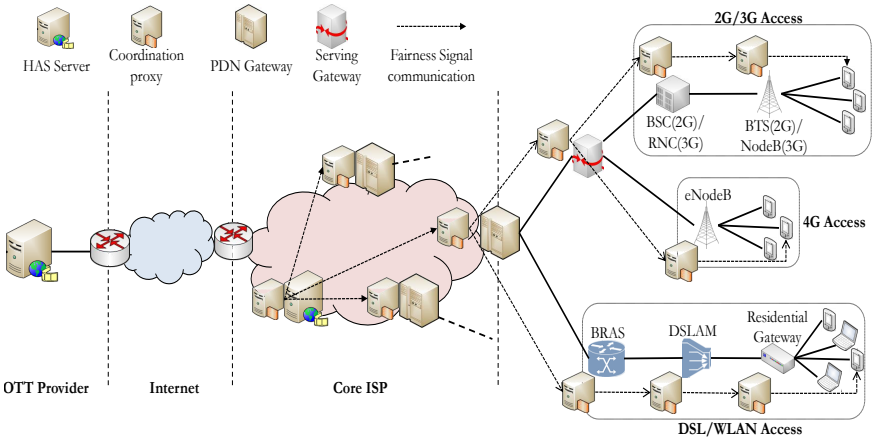


Figure 3.1: Schematic representation of a communication network. The possible locations of the coordination proxies are shown. If the streaming service is offered by an Over-The-Top (OTT) provider, the proxy located at the HAS Server in the core ISP network has to be moved to the router connecting the core ISP network with the Internet.

nication is needed among clients and consequently no significant overhead is introduced. Moreover, no client-to-proxy communication is required. The proxies are *transparent* to the clients, as the clients only need to know how to access the fairness signal but not how it is created. Second, the computation and delivery of the fairness signal do not negatively affect the behavior of existing clients. Third, the approach is robust toward proxy failure, as the clients can also operate without the fairness signal.

As far as the coordination proxies positioning is concerned, the proxies should be located at the main aggregation points of the network, in order to monitor the links where a bottleneck can occur. Potential bottlenecks can be identified by analysing the underlying network architecture or at runtime by monitoring link conditions (e.g., if the traffic exceeds a certain percentage of the link capacity, a coordination proxy can automatically become active). Since network operators have full control of their delivery infrastructure, they can easily identify which are the most sensible paths in their networks where a bottleneck could occur. This way, they can perform an initial placement of the coordination proxies on network nodes. Note that this assumption does not impact the flexibility of our solution, since in a real scenario the network architecture is given and does not significantly change over time. Furthermore, as coordination proxy functionalities can be implemented via software, proxies can be flexibly relocated in case network conditions consistently change over time. Moreover, coordination proxies can be placed liberally on network nodes, without negatively impacting the fairness signal computation even if a bottleneck does not occur. In this case, a proxy only receives

the fairness signal from its parent node and forwards it to its child proxies, without performing any operation on it. In other words, if a bottleneck does not occur, the considered proxy becomes transparent with respect to the computation of the fairness signal. As an example of coordination proxies placement, Figure 3.1 depicts a schematic representation of a communication network architecture based on the Evolved Packet Core (EPC), as described by Release 11 of the 3rd Generation Partnership Project (3GPP) [25]. The EPC represents a unified core network for the convergence of both 3GPP (as 2G, 3G and 4G) and non-3GPP (as WLAN or DSL) access networks, but it is worth noting that our approach can be applied to other network topologies as well. Depending on whether the streaming service is offered by the network operator or an Over-The-Top (OTT) provider, the HAS server can be located inside the core ISP network or in a data center connected to the core ISP network through the Internet, respectively. In the latter case, a coordination proxy is located at the edge between the Internet and the core ISP network. In 2G and 3G access networks a base station and a base station controller manage the radio interface. In the 4G case these functionalities are carried out by the same node. A DSLAM concentrates the traffic coming from WLAN and DSL access networks and forwards it to the Broadband Remote Access Server (BRAS). A Packet Data Network (PDN) Gateway aggregates the traffic belonging to 3GPP and non-3GPP access networks and directs it into the core ISP network. The aforementioned nodes can provide coordination proxy functionalities, since they aggregate traffic at different network levels and can detect the presence of congested links. It is worth noting that the proposed system can be extended to take into account the presence of caches. In this case, the proxies have to compute a separate fairness signal for each of the considered delivery nodes (e.g., a generic HAS server or a cache). This is necessary, as the different delivery nodes might be associated with different end-to-end fair bandwidth estimations. The proxies are in this case responsible for embedding the right fairness signal in the HTTP header returned to the client while downloading the segment, based on the delivery node that is currently offering the video content. Consequently, our approach can be combined with any existing HAS-optimized caching scheme.

An important aspect of our framework is how the bandwidth estimation process is carried out by the proxies. In this chapter, we use an implementation based on sFlow, which performs bandwidth estimation through an operation of packet sampling [26]. Proxies monitor the bandwidth for two groups of flows: HAS traffic and non-HAS traffic (i.e., the traffic generated by non-HAS clients is grouped into the same flow). When a packet is sampled, the packet header is extracted and analyzed in order to match it to a specific flow. A packet belongs to the HAS flow if the IP source address matches that of the HAS server and the TCP source port is equal to 80 (the standard port for HTTP connections). Otherwise, the packet is considered part of the non-HAS traffic flow.

*Algorithm 1: FINEAS client heuristic. The symbol * indicates the parameters belonging to the heuristic.*

Require: *videoBitRate*, vector containing the bitrate of each quality level of the video
maxAvailableQuality, highest available quality level of the video
segmentDuration, duration of the video segments, in seconds
bufferSize, size of the video player buffer, expressing the maximum amount of video that can be stored, in seconds
*qualityWindow**, time window stating how many quality level samples should be kept in memory, in seconds
*bufferMin**, panic threshold of the buffer level, in seconds
*bufferPercentage**, target percentage of the buffer size
α*, global utility function weight
Ensure: *nextQL*, the next quality level to request

$bandwidth =$ getLocalPerceivedBandwidth()
$bufferFillingLevel =$ getCurrentBufferFillingLevel()
$bufferTarget = bufferSize \times bufferPercentage$
$qualityHistoryVector =$ updateQualityHistoryVector($currentTime$,$qualityWindow$)
$averageQuality =$ computeAverageQuality($qualityHistoryVector$)
$fairnessSignal =$ getFairnessSignal()

$fairnessQuality = 1$
for $ql = 1$ **to** $maxAvailableQuality - 1$ **do**
  if $videoBitRate(ql) \leq fairnessSignal < videoBitRate(ql+1)$ **then**
    $fairnessQuality = \frac{fairnessSignal + ql \times videoBitRate(ql+1) - (ql+1) \times videoBitRate(ql)}{videoBitRate(ql+1) - videoBitRate(ql)}$
  end if
end for
if $fairnessSignal \geq videoBitRate(maxAvailableQuality)$ **then**
  $fairnessQuality = maxAvailableQuality$
end if
if $bufferFillingLevel \leq bufferMin$ **then**
  $nextQL = 1$
else
  $maxDownloadableQuality = maxAvailableQuality$
  for $ql = 1$ **to** $maxAvailableQuality$ **do**
    $estimatedDownloadTime = (videoBitRate(ql) \times segmentDuration) / bandwidth$
    $estimatedBufferLevel = bufferFillingLevel - estimatedDownloadTime + segmentDuration$
    if $estimatedBufferLevel \leq bufferMin$ **then**
      $maxDownloadableQuality = ql - 1$
      break
    end if
  end for
  $globalUtilityMax = -1000$
  for $ql = 1$ **to** $maxDonwloadableQuality$ **do**
    $estimatedDownloadTime = (videoBitRate(ql) \times segmentDuration) / bandwidth$
    $estimatedBufferLevel = bufferFillingLevel - estimatedDownloadTime + segmentDuration$
    $qoeUtility = - \mid ql - maxDownloadableQuality \mid - \mid ql - averageQuality \mid - \mid estimatedBufferLevel - bufferTarget \mid$
    $fairnessUtility = - \mid ql - fairnessQuality \mid$
    $globalUtility = (1 - \alpha) \times fairnessUtility + \alpha \times qoeUtility$
    if $globalUtility \geq globalUtilityMax$ **then**
      $globalUtilityMax = globalUtility$
      $nextQL = ql$
    end if
  end for
  return nextQL
end if

3.3.2 Client-Side Rate Adaptation

The FINEAS heuristic is executed by the client when a new segment has been downloaded, and before the request of a new one. The goal of the client is to

select the most appropriate quality level in order to maximize QoE and achieve fairness. A detailed description of the heuristic is provided in Algorithm 1. We indicate all the parameters belonging to our heuristic with the symbol *. In Section 3.4, an extensive evaluation of the impact of these parameters on the algorithm's performance is reported. The FINEAS heuristic can be subdivided into four main parts we describe in the following.

Input collection (lines 1-15). First, the client obtains the local perceived bandwidth, the video player buffer filling level *bufferFillingLevel* and the buffer target *bufferTarget* (lines 1-3). The perceived bandwidth is the bandwidth measured during the download of a segment and is computed as the ratio between the segment size and the segment download time. *bufferFillingLevel* is the amount of video currently stored in the video player buffer, expressed in seconds. It depends on the number of segments currently stored in the buffer and their duration. *bufferTarget* represents the target for the buffer filling level, expressed in seconds. The client requests the next segment in order to maintain the resulting buffer filling level close to *bufferTarget*. This value is set by the parameter *bufferPercentage*, which is expressed as a percentage of the total buffer size. As an example, given *bufferSize* and *bufferPercentage* are equal to 10 seconds and 80% respectively, the desired buffer target would be equal to $10 \times 0.8 = 8$ seconds. Next, the client updates the vector *qualityHistoryVector*, which contains the quality levels requested in the last *qualityWindow* seconds, and computes the average requested quality over this time window (lines 4-5). The fairness signal is then extracted from the HTTP header of the downloaded segment (line 6). The fairness signal represents the fair bandwidth share per client as computed by the coordination proxies. A client should request the quality level whose bitrate is closest to the fairness signal, in order to fairly share resources with the other clients. Consequently, a reference quality level *fairnessQuality* is obtained starting from the *fairnessSignal* and the bitrates of the video (lines 7-15). *fairnessQuality* is then used as the target quality level to request in order to obtain perfect fairness among clients. Note that this mapping produces a real value in the interval [*1;maxAvailableQuality*], since in general there is not an exact match between the fairness signal and the bitrates of the video. The client identifies the nearest lower (*ql*) and nearest higher (*ql+1*) bitrates compared to *fairnessSignal* (line 9). *fairnessQuality* is computed on line 10. This operation corresponds to finding the fitting line passing through the points *a=(ql,videoBitRate(ql))* and *b=(ql+1,videoBitRate(ql+1))*, and using the obtained equation to find the coordinate of point *c=(fairnessQuality, fairnessSignal)*, where the unknown variable is *fairnessQuality*. Particularly, by substituting points *a* and *b* into the equation *y=mx+k* and solving it with respect to *m* and *k*, the fitting line can be found. *m* and *k* are equal to $videoBitRate(ql+1) - videoBitRate(ql)$ and $(ql+1) \times videoBitRate(ql) - ql \times videoBitRate(ql+1)$, respectively. By substituting the coordinate of point *c* into the aforementioned equation and solv-

ing it with respect to *fairnessQuality*, the formula expressed on line 10 is obtained. If *fairnessSignal* is greater than the highest available bitrate, *fairnessQuality* is assigned to the highest available quality level (lines 13-14).

Buffer status control (lines 16-17). If the buffer level *bufferFillingLevel* is below the panic threshold *bufferMin*, the lowest quality level is directly requested. This way, the client tries to timely react to a situation where the risk of video freezes is high.

Highest downloadable quality level identification (lines 19-27). Otherwise, the heuristic identifies the highest quality level *maxDownloadableQuality* that can actually be downloaded. For each of the available quality levels, the client computes an estimate of the download time (line 21). We refer here to an estimate as the computation is performed using the local perceived bandwidth, which could change in the future. The download time is used to compute an estimate of the buffer level when the download of the next segment to request will be completed, called *estimatedBufferLevel* (line 22). The client identifies the first quality level *ql* such that *estimatedBufferLevel* is lower than the panic threshold *bufferMin* (line 23). This means that if *ql* or a higher quality level is requested, the buffer level is likely to be below the panic threshold at the end of the segment download. In order to avoid a possible freeze or the request of the lowest quality level at the next step, the highest quality level that can be downloaded (i.e., *maxDownloadableQuality*) is then set to *ql*-1 (line 24).

Final quality level selection (lines 28-39). The client finally requests the quality level maximizing a global utility, given by the weighted sum of a QoE term and a fairness term. The QoE term (line 32) encompasses the three most important factors affecting the user experience in HAS: (i) segment quality, (ii) quality switching and (iii) freezes. Segment quality is maximized by requesting the highest downloadable quality level (i.e., $-|ql\text{–}maxDownloadableQuality|$), switching is minimized by reducing the deviation from the average requested quality within the quality window (i.e., $-|ql\text{–}averageQuality|$), and freezes are minimized by keeping the buffer level as close to the buffer target level as possible (i.e., $-|estimatedBufferLevel\text{–}bufferTarget|$). Moreover, by keeping the buffer level close to a target value smaller than the maximum buffer size, we can reduce the on-off download pattern typical in HAS and guarantee a more continuous flow of GET requests from the client. The fairness term is maximized by requesting the quality level closest to the *fairnessQuality* (line 33). This way, the client is driven to request the best quality level to fairly share bandwidth with the other clients. For each quality from 1 to *maxDownloadableQuality*, the global utility is computed as the linear combination of the QoE and fairness terms (line 34). The selected quality level is the quality that maximizes this global utility (lines 35-38). A parameter α weighs the importance of the QoE and fairness components on the global utility. For $\alpha \rightarrow 1$, the heuristic tends to only maximize the QoE of the client, without

Algorithm 2: In-network computation performed at the coordination proxy P.

Require: *childProxies*, vector containing the child coordination proxies of proxy P
controlledClients, vector containing the number of HAS clients controlled by each proxy in *childProxies*
Ensure: *fairnessSignal*, vector containing the fairness signals to forward to the proxies in *childProxies*

1: $parentFairnessSignal =$ getParentFairnessSignal()
2: $unusedBandwidth = 0,\ entitledClients = 0$
3: **for** i **in** $childProxies$ **do**
4: $estimatedBandwidth \leftarrow$ getEstimatedBandwidthForHASTrafficTowardsChildProxy(i)
5: $maxFairnessSignal(i) = estimatedBandwidth/controlledClients(i)$
6: **if** $maxFairnessSignal(i) \leq parentFairnessSignal$ **then**
7: $unusedBandwidth+=(parentFairnessSignal - maxFairnessSignal(i)) \times controlledClients(i)$
8: **else**
9: $entitledClients+=controlledClients(i)$
10: **end if**
11: **end for**
12: $childProxies =$ sortIncreasing($childProxies, maxFairnessSignal$); ##sort proxies in *childProxies* in increasing order of associated *maxFairnessSignal*##
13: **for** i **in** $childProxies$ **do**
14: **if** $maxFairnessSignal(i) \leq parentFairnessSignal$ **then**
15: $fairnessSignal(i) = maxFairnessSignal(i)$
16: **else**
17: $bandwidthPerClient = unusedBandwidth/entitledClients$
18: $fairnessSignal(i) = \min(parentFairnessSignal + bandwidthPerClient, maxFairnessSignal(i))$
19: $unusedBandwidth-=(fairnessSignal(i) - parentFairnessSignal) \times controlledClients(i)$
20: $entitledClients-=controlledClients(i)$
21: **end if**
22: **end for**
23: **return** $fairnessSignal$

taking into account fairness. For $\alpha \to 0$, only fairness is optimized. The influence of this parameter is investigated in Section 3.4.3.

The computational complexity of our heuristic can be estimated as $O(n)$, with *n* the number of quality levels, given the complexity of the loops on lines 8, 20 and 29 is $O(n)$. The algorithm has been implemented on a Dell Latitude E5530 running Ubuntu 12.04 LTS 64-bit, Intel Core i5-3320M CPU @ 2.60GHz processor and 8 GB of memory to evaluate scalability. Even with 10000 available quality levels, the quality selection can still be performed in less than 6 milliseconds.

3.3.3 Fairness Signal Computation

The hierarchical system of coordination proxies has the fundamental role to help the clients in achieving fairness, by periodically computing the fairness signal. This value represents the theoretical fair bandwidth allocation of the clients belonging to a certain network and is computed taking into account all the clients

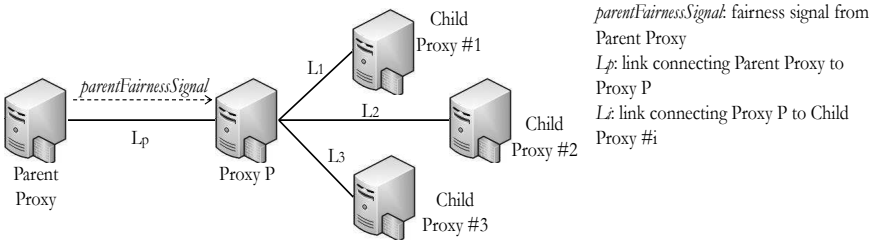


Figure 3.2: Schematic representation of the coordination proxy architecture

streaming video (even those belonging to other networks). Algorithm 2 describes the operations performed by a generic coordination proxy P to compute its estimate of the fairness signal.

As depicted in Figure 3.2, proxy P is connected to a parent proxy and to a set of child proxies, each controlling a certain number of clients as indicated in the vector *controlledClients*. From the parent node, P receives the fairness signal *parentFairnessSignal* (line 1). This value represents the fair bandwidth share for each client controlled by P, as computed by the parent node. Proxy P aims to fairly redistribute the bandwidth represented by *parentFairnessSignal* to its child proxies. In case P is the root of the system, *parentFairnessSignal* has to be assigned to infinite, as we assume there are no bottlenecks before the root proxy.

For each child proxy i, P obtains the estimated bandwidth for HAS traffic on the link connecting P to proxy i (link L_i in Figure 3.2). Next, it computes the fair bandwidth share on this link, i.e., the ratio between the estimated bandwidth on L_i and the number of clients controlled by proxy i (line 5). This value, indicated as *maxFairnessSignal(i)*, represents the maximum fairness signal achievable for proxy i. Proxy P then checks whether L_i or L_P, the link connecting the parent proxy to P, is the actual bottleneck for proxy i. Particularly, if the fair bandwidth per client on link L_i (i.e., *maxFairnessSignal(i)*) is smaller than that on link L_p (i.e., *parentFairnessSignal*), the bottleneck is represented by link L_i. Otherwise, if *maxFairnessSignal(i)*>*parentFairnessSignal*, L_p acts as bottleneck for proxy i. When L_i represents the bottleneck (line 6), the clients controlled by proxy i are not able to use all the bandwidth indicated by *parentFairnessSignal*. Particularly, the unused bandwidth is equal to the difference between *parentFairnessSignal* and *maxFairnessSignal(i)*, multiplied by the number of clients controlled by proxy i (line 7). The leftover bandwidth *unusedBandwidth* can be redistributed to the clients whose bottleneck is represented by link L_p. When L_p is the bottleneck, the number of clients entitled to accept part of the exceeding bandwidth is incremented by the number of clients controlled by proxy i (line 9).

The final fairness signal for each child proxy is computed on lines 13-22. Here, proxy P redistributes the exceeding bandwidth to all the entitled clients. If L_i is

the bottleneck for child proxy *i*, i.e., no extra bandwidth can be assigned, the final fairness signal is assigned to *maxFairnessSignal(i)* (lines 14-15). Otherwise, the surplus bandwidth per client *bandwidthPerClient* is computed as the ratio between the unused bandwidth and the number of entitled clients (line 17). Next, the final fairness signal for proxy *i* is assigned as the sum between the parent fairness signal and the surplus bandwidth per client (line 18). The *min* operation performed on line 18 avoids that *fairnessSignal(i)* exceeds *maxFairnessSignal(i)*. The amount of unused bandwidth is then decremented by the amount assigned at the previous step (line 19). Also, the amount of entitled clients is decremented by the number of clients controlled by proxy *i* (line 20).

As an example, we consider the scenario in Figure 3.2, where proxy *P* is connected to three child proxies. We assume each child proxy controls 10 HAS clients (i.e., *controlledClients(i)*=10 for *i*=1,2,3), and the estimated bandwidth for HAS traffic on links L_1, L_2 and L_3 is 10, 20 and 35 Mbps, respectively. *parentFairnessSignal* is 2 Mbps. In this case, the variable *maxFairnessSignal(i)* for each proxy is equal to 1, 2 and 3.5 Mbps, respectively. This entails that L_1 represents the main bottleneck for proxy #1, as *maxFairnessSignal(1)*$\leq$*parentFairnessSignal*. The unused bandwidth is computed as on line 7 and is equal to *unusedBandwidth*=10 Mbps. This is not the case for proxy #3, as *maxFairnessSignal(3)* is greater than *parentFairnessSignal*. Consequently, the number of entitled clients is incremented by 10 clients. The fairness signal for proxy #1 and #2 is assigned directly to 1 and 2 Mbps, respectively, as both *maxFairnessSignal(1)* and *maxFairnessSignal(2)* are smaller or equal to *parentFairnessSignal*. On the contrary, proxy *P* can assign the exceeding bandwidth to the clients controlled by proxy #3. The variable *bandwidthPerClient* is equal to 1 Mbps and the final signal is equal to *parentFairnessSignal*+*bandwidthPerClient*=3 Mbps, as shown on line 18.

The rationale behind the sort operation performed on line 12 is to reduce the computational complexity of the heuristic. The complexity of the two *for* loops on lines 3 and 13 is $O(n)$, while that of the sort operation is $O\left(n \times log(n)\right)$, with *n* the length of vector *childProxies*. Consequently, the total computational complexity is $O\left(n \times log(n)\right)$, which is known to scale well even for large values of n. If the vector *childProxies* is not sorted, the computation on lines 13-22 has to be performed using two nested *for* loops, for a total computational complexity of $O(n^2)$. In order to evaluate complexity, Algorithm 2 has been implemented on a Dell Latitude E5530 running Ubuntu 12.04 LTS 64-bit, Intel Core i5-3320M CPU @ 2.60GHz processor and 8 GB of memory. We incremented the number of child coordination proxies from 2 to 10000 and measured the time needed to compute the fairness signal. Even for 10000 child proxies, the fairness signal computation is carried out in less than 0.032 seconds on a general purpose device. Moreover, it is worth noting that the fairness signal computation is decoupled from the actual segment delivery to the clients. When a proxy receives a segment to forward, it

just has to embed its current estimate of the fairness signal into an HTTP header. The actual update of the fairness signal can be considered as an independent and parallel process occurring periodically within the system of coordination proxies.

The fairness signal computation period, i.e., the communication interval among the proxies, is an important parameter of our algorithm. A small period entails that the fairness signal closely follows bandwidth variations, at the price of an increased computational and communication overhead. On the contrary, a high period reduces overhead but negatively affects the significance of the fairness signal. The impact of this parameter on the performance of our solution is evaluated in Section 3.4.

In the fairness signal computation presented in Algorithm 2, it is assumed that the link capacity is shared equally among the HAS clients. This assumption might not hold in wireless networks, where the capacity is not only shared among users, but it also depends on the specific user location and mobility (i.e., the quality of the wireless channel or RSSI). This information could be used by the last coordination proxy (see Figure 3.1) to refine the final estimation of the fairness signal. Wireless channel information can indeed be exploited to infer the optimal transmission rate of a client, which would allow to perform a better redistribution of the fairness signal among the HAS clients in the wireless network.

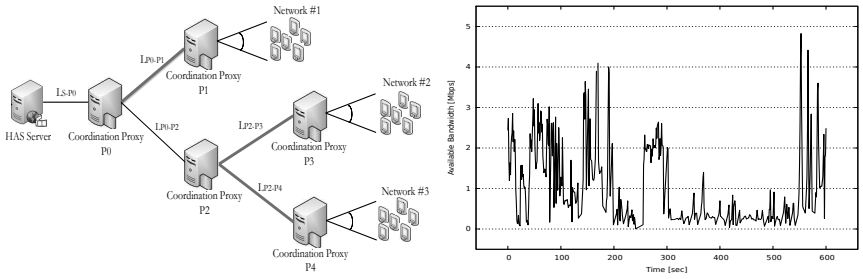
3.4 Performance Evaluation Results

3.4.1 Experimental Setup

An NS-3-based simulation framework has been used to evaluate our multi-client framework [19]. The video streamed is *Big Buck Bunny*, composed of 299 segments, each 2 seconds long and encoded at 7 different quality levels: 300, 427, 608, 806, 1233, 1636, 2436 kbps (the lowest quality level is indicated with 1). The Network Simulation Cradle[1] has been enabled for all the results, in order to provide a realistic implementation of the TCP protocol. Unless otherwise stated, the buffer size for each client is equal to 5 segments, or 10 seconds.

The simulated network topology is shown in Fig. 3.3a, where the position of the coordination proxies is reported. The root proxy is co-located with the HAS Server (not shown in Fig. 3.3a). In order to give an extensive evaluation of the FINEAS heuristic, we simulate 50 episodes of the video trace and average the results over the 50 runs. The capacity on links $L_{S\text{-}P0}$ and $L_{P0\text{-}P2}$ is kept fixed for all the experiments. A cross traffic generator introduces UDP traffic on links $L_{P0\text{-}P1}$, $L_{P2\text{-}P3}$ and $L_{P2\text{-}P4}$ (highlighted links in Figure 3.3a), in order to vary the available bandwidth for HAS traffic. A different cross traffic pattern is used for each episode, which varies each second and is scaled with respect to the number of

[1] http://research.wand.net.nz/software/nsc.php



(a) (b)

Figure 3.3: Simulated topology (a) and an extract of the used bandwidth pattern (b).

clients. An episode is defined as a single simulation run. As far as the cross traffic pattern is concerned, we use an open-source dataset collected on a real 3G/HSDPA network [27]. The available bandwidth for one client fluctuates between 202 bps and 6335 kbps, with an average of 2087 kbps and a standard deviation of 1314 kbps. An example of the bandwidth pattern is shown in Figure 3.3b. Regarding the bandwidth estimation at the coordination proxies, the sFlow packet sampling rate has been set to 1 packet sampled every 1000 for all the experiments. As far as the fairness signal is concerned, it is added as an HTTP header field by the proxies and returned to the clients when delivering the next segment.

In order to provide an extensive benchmark of the FINEAS algorithm, we compare our results to those obtained using four other HAS clients. Particularly, we choose a proprietary HAS client, the MSS[2] client, the Q-Learning-based client described by Claeys et al. [11] and the client developed by Miller et al. [28]. We also studied the performance of the FESTIVE algorithm, one of the first algorithms developed to explicitly deal with a multi-client scenario [13]. As far as the performance evaluation is concerned, fairness is computed as the standard deviation of clients' QoE. Unless otherwise stated, all the simulated clients start streaming the video at the same time.

3.4.2 QoE Model

In this section, we define the QoE model used to evaluate the proposed framework. We use a metric in the same range of the Mean Opinion Score (MOS), that can be computed as in Eq. 3.1 [11, 29]:

$$QoE_j^i = 5.67 \times \frac{\bar{q}_j^i}{{q_{max}}_j^i} - 6.72 \times \frac{\hat{q}_j^i}{{q_{max}}_j^i} + 0.17 - 4.95 \times F_j^i \qquad (3.1)$$

[2]https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

Table 3.1: Overview of evaluated parameter configuration

Parameter	*Evaluated values*	*Parameter*	*Evaluated values*
qualityWindow [sec]	10, 20, 30, 50, 70	bufferMin [sec]	2, 4, 6
bufferPercentage	0.4, 0.6, 0.8, 1.0	α	0, 0.2, 0.4, 0.6, 0.8, 1.0
T_{fair} [sec]	2, 4, 6		

q represents the quality levels of the video, expressed as an integer ranging from 1 (the lowest quality) to q_{max} (the highest quality). The QoE experienced by client j in network i is the linear combination of the average quality level requested $\bar{q}_j^i$, its standard deviation $\hat{q}_j^i$ (both normalized with respect to the highest available quality level ${q_{max}}_j^i$) and F_j^i, which models the influence of freezes and is computed as follows:

$$F_j^i = \frac{7}{8} \times max\left(\frac{ln(\phi_j^i)}{6} + 1, 0\right) + \frac{1}{8} \times \left(\frac{min(\psi_j^i, 15)}{15}\right) \tag{3.2}$$

ϕ_j^i and ψ_j^i are the freezes frequency and the average freeze duration, respectively. All the coefficients reported in Eq. 3.1-3.2 have been fixed according to the works by De Vriendt et al. [29] and Claeys et al. [11].

3.4.3 Parameter Analysis

In this section we investigate the impact of the parameters characterizing our HAS framework on clients' performance, and select the configuration to use in the remainder of the chapter. As explained in Section 3.3.2 and Algorithm 1, the FINEAS heuristic parameters are: *qualityWindow*, a time window stating how many quality levels have to be kept in memory, *bufferMin*, the panic buffer threshold, *bufferPercentage*, to tune the target buffer level, and α, the global utility function weight. As far as the in-network coordination is concerned (see Section 3.3.3), the only parameter to set is the fair signal computation period, named T_{fair}. In order to properly tune our HAS framework and select the best configuration, an elaborate evaluation of the parameter space has been performed. The evaluated values are reported in Table 3.1, for a total of 648 feasible configurations (note that the target buffer level needs always to be higher than the panic buffer threshold). We evaluate our solution with the setting reported in Section 3.4.1; each network containing 10 clients streaming video at the same time. The capacity of links $L_{S\text{-}P0}$ and $L_{P0\text{-}P2}$ is set to 60 Mbps and 40 Mbps respectively, while the bandwidth on links $L_{P0\text{-}P1}$, $L_{P2\text{-}P3}$ and $L_{P2\text{-}P4}$ is variable and has an average of 20 Mbps and a standard deviation of 13 Mbps over the 50 simulated episodes. This simulation setup allows us to have a clear understanding of the impact of the different parameters on the performance of our solution. Depending on the available bandwidth on links $L_{P0\text{-}P1}$, $L_{P2\text{-}P3}$ and $L_{P2\text{-}P4}$, the actual bottlenecks for the three networks dynamically

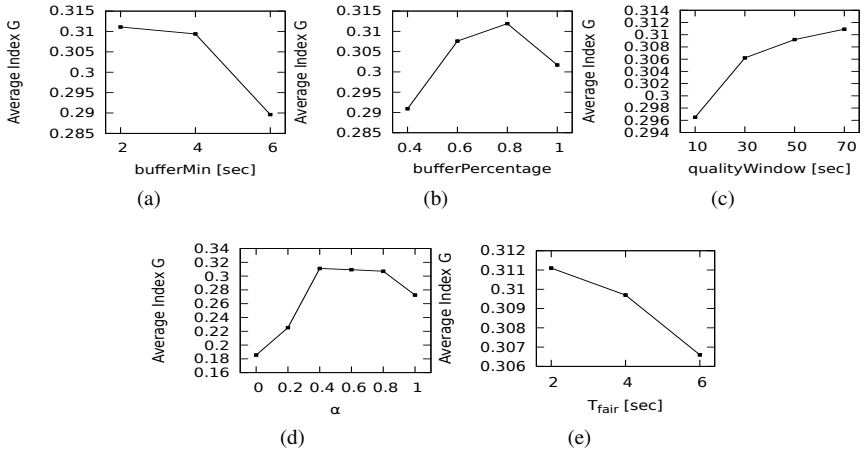


Figure 3.4: Analysis of the parameter influence

change. This way, we can explore a wide range of network configurations, where the three networks mutually influence each other (due to the bandwidth limitations on links $L_{S\text{-}P0}$ and $L_{P0\text{-}P2}$) or not. Other topologies have been investigated, but the results do not significantly change and are omitted due to space constraints.

The performance evaluation is conducted on the basis of the achieved QoE and fairness. Particularly, we express fairness as the standard deviation of clients' QoE. We introduce a metric G_k to evaluate the overall performance of the analyzed configurations, defined as in Eq. 3.3:

$$G_k = \frac{1}{N}\sum_{i=1}^{N} J_k^i - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(J_k^i - \frac{1}{N}\sum_{i=1}^{N} J_k^i\right)^2} \quad \text{with} \quad J_k^i = \overline{QoE}_k^i - \widehat{QoE}_k^i \tag{3.3}$$

N represents the number of networks containing HAS clients streaming video. J_k^i represents the performance of network *i* during the *k-th* episode and is the linear combination of $\overline{QoE}_k^i$, the average QoE computed over the whole group of clients belonging to network *i* for episode *k*, and $\widehat{QoE}_k^i$, its standard deviation. This way, J_k^i encodes the fairness objective for network *i*, i.e., maximizing QoE while improving fairness. G_k is used to evaluate the overall performance of the analyzed configuration and is given by the linear combination of the average J_k^i over all networks and of the negative of its standard deviation. Our aim is to select a configuration that allows to maximize G_k, in order to achieve the highest possible performance for every network while keeping the deviation among them as low as possible. For every configuration, the index *G* has been computed as the

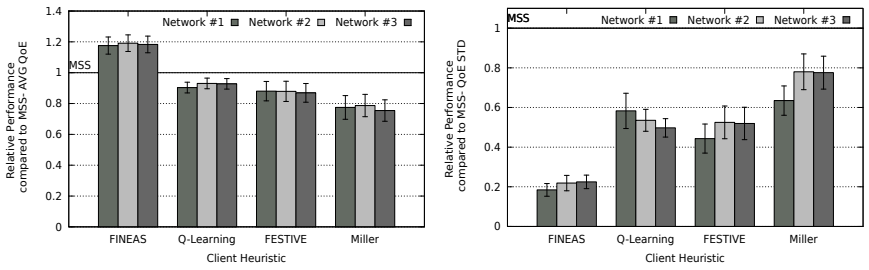
average of G_k over the 50 simulated episodes. For each evaluated value of each parameter, the average G of the five best configurations containing the evaluated value has been calculated. Using more than 5 values does not significantly affect the outcome of this analysis. The results are shown in Figure 3.4. Small values of the panic threshold *bufferMin* lead to better performance, as they avoid to often request the lowest quality level. The increased risk for video freezes caused by a low panic threshold is counterbalanced by using higher values of the buffer percentage *bufferPercentage*, which led to higher buffer targets. A trade-off is present in this case: a high percentage reduces the risk of freezes but also the possibility to request a high quality level. The best performance is reached for a value of 0.8. As far as the *qualityWindow* is concerned, it appears clearly that the more quality levels stored, the better. This entails that the client has a more comprehensive vision of the actions taken in the past, and can choose the next quality level accordingly. The parameter α states to what extent the clients follow the indication given by the fairness signal. The best value is obtained at 0.4, meaning that the two opposite objectives of maximizing clients' QoE and achieving fairness are well-balanced. Strictly following the fairness signal (i.e., values of α close to zero) leads to perfect fairness but bad performance in terms of QoE, because of the frequent switches that would occur. This result also highlights that it is more convenient to incorporate the fairness signal into the adaptation heuristic rather than using it directly to enforce the quality selection process. As expected, a small value of the fairness signal computation period T_{fair} leads to the best performance, as the in-network computation can quickly follow the bandwidth variations. The introduced communication overhead can be kept minimal by adding an HTTP header to transport the fairness signal.

Based on this analysis, the configuration with *qualityWindow* equal to 70 seconds, *bufferMin* to 2 seconds, *bufferPercentage* to 0.8, α to 0.4 and T_{fair} to 2 seconds has finally been chosen. It is worth noting that this configuration is also the best overall, thus showing the validity of the parameter analysis.

3.4.4 Variable Bandwidth Scenario

After the parameter analysis, we evaluate our solution with an increased number of clients. A fixed line scenario has also been evaluated, but the results are omitted due to space limits. In a fixed line scenario, all the simulated heuristics are able to guarantee a high QoE, but only the FINEAS heuristic is able to guarantee fairness.

The same simulation setting as in Section 3.4.1 is used, with 30 clients per network streaming a video at the same time. In this case, links $L_{S\text{-}P0}$ and $L_{P0\text{-}P2}$ are fixed to 180 Mbps and 120 Mbps respectively, while the bandwidth on links $L_{P0\text{-}P1}$, $L_{P2\text{-}P3}$ and $L_{P2\text{-}P4}$ is variable and presents an average of 60 Mbps and a standard deviation of 40 Mbps over the 50 simulated episodes. In this scenario,



(a) (b)

Figure 3.5: Comparison between the different clients, from a QoE perspective, for a variable bandwidth scenario. Each network contains 30 clients streaming video. The graphs report the relative performance of the considered clients in terms of (a) average QoE and (b) its standard deviation compared to MSS. The standard deviation of clients' QoE is used as fairness metric.

the three networks can mutually influence their performance, as the links $L_{S\text{-}P0}$ and $L_{P0\text{-}P2}$ act as common bottlenecks. It is fundamental to stress that the variable bandwidth pattern has been collected on a real 3G/HSDPA network, as described in Section 3.4.1. We consider the MSS as reference client and compute, for each simulated episode and for each network, the ratio between the average QoE of the analyzed client and that of the MSS. Figure 3.5a reports the average value over the 50 episodes of this ratio, together with the confidence intervals at 95%. Figure 3.5b reports the average value over the 50 episodes of the ratio between the QoE standard deviation of the MSS algorithm and that of the analyzed client, together with the confidence intervals at 95%. The QoE standard deviation is used as fairness metric. Our solution is able to increase the average QoE by almost 20% for each of the three networks and to improve fairness with almost 80% when compared to MSS. Also the FESTIVE and Miller clients improve fairness, but consistently reduce the average QoE. This entails that the final QoE at the clients with these two heuristics is lower than that obtained using the MSS heuristic. These results show the sub-optimality of these two heuristics in case of a variable bandwidth, caused by frequent quality switches and video freezes. As far as the Q-Learning client is concerned, it improves fairness by about 50% with respect to MSS, but with a loss of 8-10% in terms of average QoE. This negative behavior is mainly due to the mutual influence among the learning processes of the clients and the uncoordinated nature of Q-Learning [30]. When a client selects a certain quality level, it uses a portion of the shared bandwidth. This decision has an impact on the performance of the other clients and thus also on their learning process. Since the clients do not share any information among each other, this leads to a sub-optimal quality adaptation policy.

Table 3.2: Performance summary in the variable bandwidth scenario, in terms of quality components. The average value over the 50 episodes is reported, together with the confidence interval at 95%.

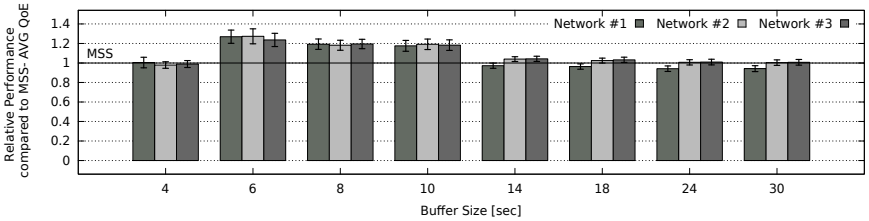
		MSS	*FINEAS*	*Q-Learning*	*FESTIVE*	*Miller*
Network 1	QoE	2.96±.18	3.41±.15	2.61±.07	2.64±.26	2.35±.32
	$\bar{q}_j^i$	4.51±.19	5.19±.13	3.69±.03	5.60±.14	5.70±.15
	$\hat{q}_j^i$	0.91±.08	0.98±.06	0.55±.03	1.23±.04	1.51±.09
	Bitrate [Mbps]	1.01±.08	1.30±.05	0.74±.01	1.47±.05	1.51±.06
	Freeze [sec]	0.08±.02	0.07±.02	0.07±.04	4.21±1.16	3.19±.59
	Freeze number	0.07±.01	0.05±.02	0.06±.03	3.42±.84	3.40±.49
Network 2	QoE	2.76±.15	3.25±.16	2.52±.08	2.44±.23	2.18±.24
	$\bar{q}_j^i$	4.21±.16	5.16±.15	3.68±.04	5.14±.11	5.24±.12
	$\hat{q}_j^i$	0.83±.07	1.12±.06	0.62±.04	1.14±.03	1.38±.06
	Bitrate [Mbps]	0.89±.06	1.29±.06	0.73±.01	1.28±.04	1.32±.05
	Freeze [sec]	0.08±.02	0.09±.03	0.10±.03	3.31±.87	2.66±.48
	Freeze number	0.08±.01	0.07±.02	0.11±.02	2.95±.71	2.99±.39
Network 3	QoE	2.78±.15	3.25±.16	2.52±.07	2.44±.22	2.11±.24
	$\bar{q}_j^i$	4.21±.16	5.16±.15	3.63±.04	5.14±.11	5.23±.12
	$\hat{q}_j^i$	0.81±.07	1.12±.06	0.57±.03	1.14±.03	1.40±.05
	Bitrate [Mbps]	0.89±.06	1.29±.06	0.73±.01	1.28±.04	1.31±.05
	Freeze [sec]	0.08±.03	0.09±.03	0.11±.03	3.35±.89	2.86±.51
	Freeze number	0.07±.01	0.07±.02	0.11±.02	2.91±.69	3.15±.39

Table 3.2 summarizes the results for this scenario. We compute, for each episode and for each network, the average of clients' QoE, $\bar{q}_j^i$, $\hat{q}_j^i$, requested bitrate, freeze time and freeze number. $\bar{q}_j^i$ and $\hat{q}_j^i$ represent, respectively, the average quality level requested by client *j* in network *i* and the standard deviation from this average (see Eq. 3.1). We then average these values over the 50 simulated episodes. The largest gain of the FINEAS heuristic is an increased average quality level (15% higher than MSS) and requested bitrate, without affecting the freeze duration. The FESTIVE and Miller clients reach the highest quality level, but the freeze duration, the number of freezes and quality switches increase. This means that the rate adaptation process is not performed optimally and, consequently, does not lead to the highest possible QoE. The main problem of the Q-Learning algorithm is the low average requested quality level, which strongly affects the final QoE. This behavior is a consequence of the mutual influence among the clients. When a client requests a high quality segment, it could experience a freeze due to the congestion caused by the other clients. Therefore, the clients develop a more conservative policy, where high quality levels are avoided to limit freezes. This problem is further intensified by the non-stationarity of the simulated environment.

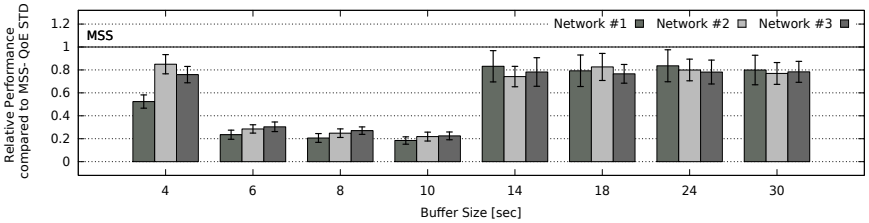
Table 3.3 reports the statistical significance of the results. For each algorithm and for each episode, we compute the average QoE and the average standard deviation over the three networks. Next, we perform a two-tail paired t-test for each pair of heuristics. Results are reported as the mean (and its standard deviation) of the average QoE and the average QoE standard deviation, over the 50 simulated

Table 3.3: Statistical significance of the average QoE and the average standard deviation of the QoE, using a two-tail paired t-testing with significance level 0.05.

	MSS	*FINEAS*	*Q-Learning*	*FESTIVE*	*Miller*
Average QoE	2.78±.50a	3.24±.55b	2.50±.38c	2.46±.58c	2.17±.60d
QoE Standard Deviation	0.69±.16a	0.13±.04b	0.32±.05c	0.31±.08c	0.46±.09d



(a) Relative performance of our solution in terms of average QoE compared to MSS.



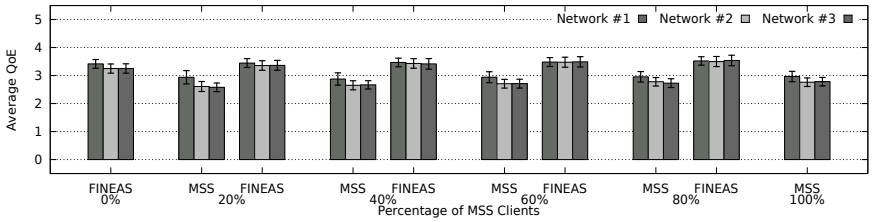
(b) Relative performance of our solution in terms of QoE standard deviaton compared to MSS.

Figure 3.6: Evaluation of the FINEAS heuristic, from a QoE perspective, for a variable bandwidth scenario with different buffer sizes. Each network contains 30 clients streaming video. The x-axis reports the buffer size, in seconds.

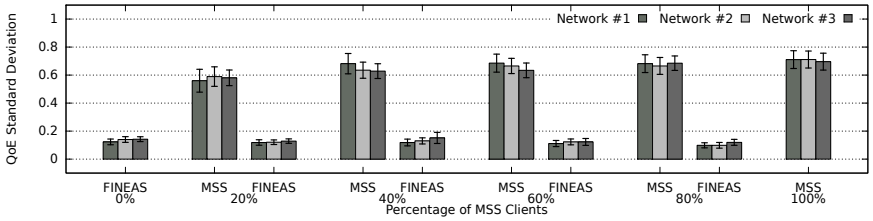
episodes. Taken two heuristics, they are statistically different if they are associated to different letters. For example, the results for the MSS and FINEAS heuristics are statistically different. Only the Q-Learning and the FESTIVE clients do not show any significant difference in performance.

3.4.5 Influence of the Buffer Size

In this section, we investigate clients' performance using different buffer sizes. Based on the results of the previous section, we decided to evaluate our solution and the MSS one, which outperforms the Q-Learning, the FESTIVE and the Miller client in terms of QoE (see Figure 3.5a). The same simulation setting of the previous section has been used, with different buffer sizes from 4 to 30 seconds. In the 4 seconds case, the panic threshold of the FINEAS heuristic has been disabled (i.e., *bufferMin* set to zero). The results of this investigation are depicted in Figure 3.6. As expected, when the buffer size increases, the differences between our solution and MSS decrease. Particularly, the MSS heuristic presents a big performance gain



(a) Average QoE of the two client groups.



(b) QoE standard deviation of the two client groups

Figure 3.7: Evaluation of the heterogeneous scenario, from a QoE perspective. Each network contains 30 clients streaming video. The x-axis reports the percentage of MSS clients on the total.

in terms of average QoE and its standard deviation when the buffer is larger than 14 seconds. Consequently, the gain brought by the FINEAS algorithm decreases. Nevertheless, the FINEAS heuristic is still able to outperform MSS by about 20% in terms of fairness. A large buffer simplifies the rate adaptation process, since it reduces the risk of freezes and thus the dependency on the current available bandwidth. When the buffer is sufficiently filled, a client can request the best quality level to maximize its QoE even though the current available bandwidth would not allow it. Despite that, a large buffer should be avoided because it causes a long delay when streaming live contents and a big memory occupation on the device. Also when the buffer is only a few seconds (4 seconds case in Figure 3.6), the performance of the FINEAS client is very close to that of MSS. Unlike the previous case, if the buffer size is very small, the client has to follow almost precisely the available bandwidth, in order to minimize the risk of freezes. Consequently, the optimization possibilities are reduced. Nevertheless, our solution always outperforms MSS from the fairness point of view, while achieving a similar or higher average QoE.

3.4.6 Client Heterogeneity

So far, we have investigated a homogeneous scenario, where all the clients are equipped with the same heuristic. In a real scenario, different client types can re-

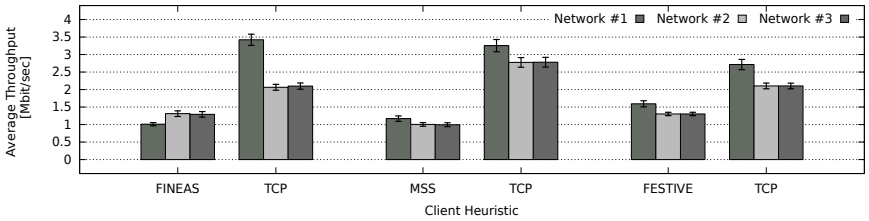
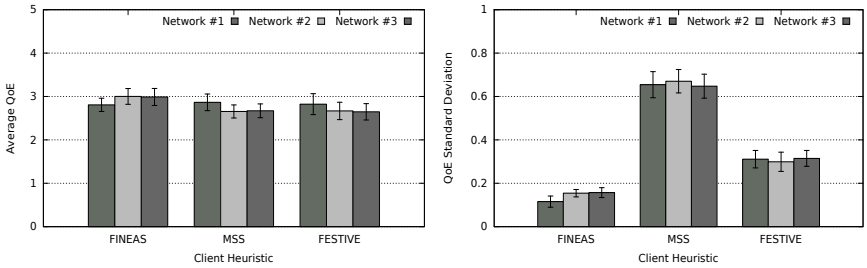


Figure 3.8: Evaluation of a scenario with concurrent TCP clients. TCP clients are 25% of the total. The x-axis reports the different heuristics, the y-axis the average throughput for each network over the 50 simulated episodes.

ceive video streams at the same time. We analyze here the interaction between two different groups of clients, one equipped with the FINEAS heuristic and the other equipped with the MSS one. The fairness signal is computed considering the totality of clients streaming video, but is analyzed by our clients only. The simulation settings are those reported in Section 3.4.4. We increase the size of the MSS group from 20% to 80% of the total, for a number of MSS clients ranging from 18 to 72; the assignment of the MSS clients to each network is made randomly at the beginning of each simulated episode. In order to evaluate the results, we compute for each simulated episode, for both client groups, the average QoE and its standard deviation. We then average these values over the 50 simulated episodes. Figure 3.7 reports the obtained results, together with the confidence intervals at 95%. The FINEAS group is always able to outperform the MSS one, with a gain in the average QoE between 20% and 30% for all the networks in all scenarios. Interestingly, the performance of the two groups remains constant even if the number of MSS clients changes. Thanks to the FINEAS heuristic and the in-network fairness signal computation, our clients are always able to select the best quality level to maximize their own QoE and optimize fairness. This optimization is carried out considering all the HAS clients, since the fairness signal is computed considering all the clients streaming video, and not only the ones equipped with our solution. This entails that our clients do not behave greedily with respect to the MSS group. The difference in performance is thus mainly due to the better rate adaptation of the FINEAS clients.

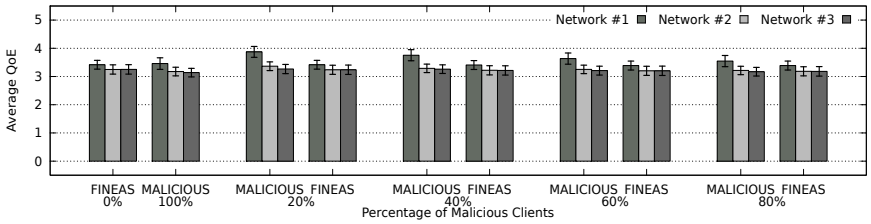
In a second set of experiments, we investigate the interaction between different HAS clients (FINEAS, FESTIVE and MSS) and generic TCP clients. The simulation settings are those reported in Section 3.4.4. The number of TCP clients is equal to 23 (i.e., 25% of the total); the assignment of the TCP clients to each network is made randomly at the beginning of each simulated episode. Each TCP client establishes a connection with the HAS Server, where a TCP sender is installed. Each client downloads the video content from the server in one continuous stream at the fixed rate of 5 Mbps. Consequently, the TCP start-up behavior is



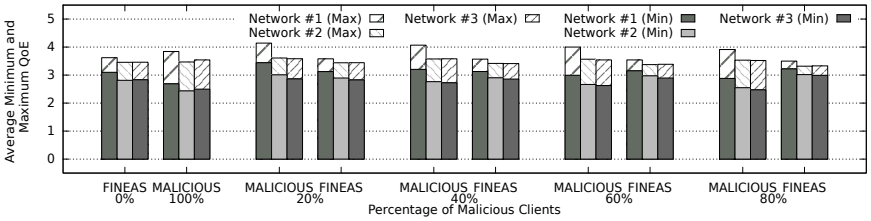
(a) (b)

Figure 3.9: Evaluation of a scenario with concurrent TCP clients, from a QoE perspective. TCP clients are 25% of the total. The x-axis reports the different heuristics, the y-axis the average QoE (a) and its standard deviation (b) for the HAS clients subgroup.

negligible compared to the total video duration and does not affect the outcome of this analysis. During each experiment, the same HAS heuristic is used. As far as the fairness signal computation is concerned, it is performed considering the HAS clients only, since we are not in control of the traffic generated by other non-HAS applications (browsing, file downloading etc.). In order to evaluate the results, we compute for each simulated episode, for both the HAS and TCP groups, the average TCP throughput. We then average this value over the 50 simulated episodes. We also compute the average QoE and its standard deviation over the 50 episodes for the HAS group. Figures 3.8 and 3.9 report the obtained results, with confidence intervals at 95%. Based on the simulated topology and the bandwidth pattern (see Section 3.4.4), each client should obtain, on average, a bandwidth of 2 Mbps. From Figure 3.8, it appears clearly that the TCP group behaves greedily with respect to the HAS group, independently from the adopted heuristic. TCP clients in Network #1, which are the closest to the HAS server and thus experience the lowest RTT, obtain the highest throughput. If we consider the results from a QoE point of view, it is possible to draw two conclusions. First, the FINEAS heuristic is still able to reach the best results both from the QoE and the fairness perspective, as one can see in Figures 3.9a and 3.9b, respectively. Particularly, the average QoE for Network #1 is the same as that of the MSS and the FESTIVE clients, while the QoE standard deviation is considerably lower in the FINEAS case. The gain achieved in terms of average QoE for Network #2 and #3 is about 13%. Second, exploited network resources being equal, the FINEAS client results in a better overall perceived video quality, i.e., is more efficient. As observed in Figure 3.8, the average TCP throughput reached by the FINEAS and FESTIVE clients for Networks #2 and #3 is almost equal and higher than that reached by MSS. Nevertheless, the QoE achieved by the FINEAS clients is considerably higher than that achieved by the two other heuristics, entailing a better utilization of network resources. The



(a) Average QoE of the two client groups. The confidence intervals at 95% are also reported



(b) Average minimum and maximum QoE of the two client groups

Figure 3.10: Evaluation of the malicious scenario, from a QoE perspective. Each network contains 30 clients streaming video. The x-axis reports the percentage of malicious clients on the total.

same consideration applies to Network #1. Even though the average throughput reached by the FINEAS clients is lower than MSS and FESTIVE, the average QoE is the same.

It is worth noting that the presence of concurrent TCP traffic might have a negative influence on the performance of the proposed system of coordination proxies. The fairness signal could induce some HAS clients to reduce their video quality in order to obtain fairness. Due to TCP interactions, the bandwidth freed by these HAS clients might be occupied by background TCP traffic. This behavior would in turn affect the estimation of HAS traffic towards child proxy nodes as computed in Algorithm 2, leading to a vicious circle. To solve this problem, a coordination proxy could take into account the evolution of the fairness signal over time. When the proxy detects that the HAS traffic on the link is decreasing over a certain period of time, which might be caused by TCP background traffic behaving greedily with respect to HAS traffic, it can tentatively increase its estimation of the fairness signal. As a result, this approach would stimulate the HAS clients requesting a higher quality, which in turn would increase again the HAS traffic in the network.

3.4.7 Malicious Clients and Proxy Failure

In this section, we analyze the robustness of our solution in presence of malicious clients and in case of a coordination proxy failure.

In the first set of experiments, we simulate the presence of malicious clients equipped with a modified FINEAS heuristic that ignores the fairness signal. The only objective of a malicious client is to maximize its own QoE, neglecting the presence of other clients and fairness. The simulation settings are those reported in Section 3.4.4. The fairness signal computation is performed considering the totality of clients streaming video, but is analyzed by the normal FINEAS clients only. We increase the size of the malicious group from 0% to 100% of the total, for a number of malicious clients ranging from 0 to 90; the assignment of the malicious clients to each network is made randomly at the beginning of each simulated episode. In order to evaluate the results, we compute for each simulated episode, for both client groups, the average QoE, its standard deviation, the minimum and the maximum QoE. We then average these values over the 50 simulated episodes. The results are reported in Figure 3.10. The first two histograms report the results for the two limit cases where the clients are all equipped with the FINEAS heuristic or are all malicious. As observed in Figure 3.10a, the two groups of clients obtain very similar average QoE. The FINEAS group obtains a QoE standard deviation half of that obtained by the malicious group (not shown in the chapter due to space limitations), as the former follows the fairness signal. More interestingly, Figure 3.10b highlights that a malicious behavior is actually detrimental for some clients belonging to the malicious group. This occurs when a malicious client would obtain a better QoE by following the fairness signal instead of ignoring it. Figure 3.10b shows the average minimum and maximum QoE obtained by the two groups, for each sub-network. The average minimum QoE for the FINEAS group is always higher or equal to the average minimum QoE of the malicious group, i.e., some malicious clients obtain a QoE lower than that they could obtain behaving fairly. Moreover, this behavior becomes more evident as the number of malicious clients increases. The only exception is represented by Network #1 in the scenario with 20% malicious clients. Nevertheless, the FINEAS group outperforms the malicious one from the fairness point of view, while achieving a similar QoE.

In the second set of experiments, coordination proxy P3 is lacking, so that clients in Network #2 do not receive the fairness signal anymore (see Figure 3.3a). This scenario represents a situation where proxy P3 is not deployed or cannot forward the fairness signal because of a fault. Figure 3.11 shows the results, obtained averaging the QoE and its standard deviation over the 50 simulated episodes. As expected, the failure of proxy P3 mainly affects the performance of Network #2 from the fairness point of view: the QoE standard deviation is almost doubled compared to the reference scenario presented in Section 3.4.4, while the average QoE is the same. This means that some clients experience a lower QoE compared to the reference scenario, as shown also in the malicious clients case. Interestingly, Networks #1 and #3 are unaffected by the failure of proxy P3, both from the QoE and the fairness points of view. This behavior can be explained considering two

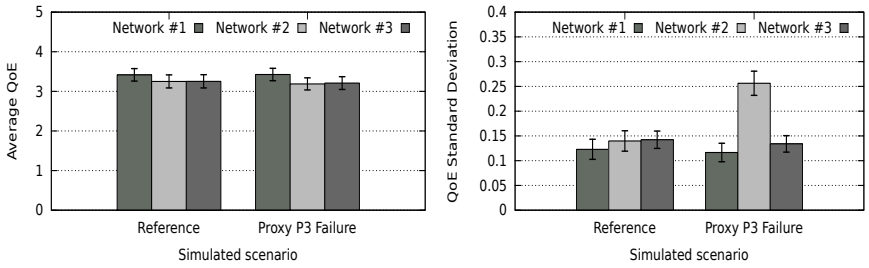


Figure 3.11: Performance evaluation, from a QoE perspective, of a scenario where Coordination Proxy P3 (see Figure 3.3a) is defective. Each network contains 30 clients streaming video. The graphs report the average QoE (a) and its standard deviation (b).

aspects. First, both networks are still provided with the fairness signal. Second, the average bandwidth consumed by Network #2 does not differ significantly from that consumed in the reference scenario. The proxy failure affects Network #2 from the fairness point of view only, while the average requested quality level, equal to 5.42, does not significantly change compared to the reference scenario (see Table 3.2). This entails that the bandwidth available to Networks #1 and #3 in case of proxy failure and in the reference scenario is similar and no differences in performance between the two scenarios are noticeable.

3.5 Conclusions

In this chapter, we presented FINEAS, a HAS heuristic able to dynamically adapt its behavior depending on network conditions, in order to obtain a high QoE. Moreover, this client is able to select the best quality level in order to achieve fairness from the QoE point of view. This was necessary as state-of-the-art rate adaptation heuristics introduce non-negligible differences in obtained quality among clients. Fairness is achieved by means of a system of intermediate nodes, called coordination proxies, in charge of collecting information on the overall network conditions. This information is then provided to the FINEAS clients, which use it to refine their quality decision process. Numerical simulations using NS-3 have validated the effectiveness of the proposed approach. Particularly, we have compared our solution with the Microsoft ISS Smooth Streaming client, a Q-Learning based client [11], a single-client heuristic [28] and the FESTIVE algorithm [13]. In the evaluated bandwidth scenarios, our multi-client HAS framework resulted in a better video quality and in a remarkable improvement of fairness, up to 20% and 80% respectively, when compared to the reference algorithms.

Acknowledgment

The research was performed partially within the iMinds V-FORCE project (under IWT grant agreement no. 130655). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme. Maxim Claeys is funded by a grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

Appendix

In this appendix, we formally present the multi-client fairness problem addressed in the chapter, based on our previous work [3]. We assume a scenario where multiple local access networks are connected to the core Internet Service Provider (ISP) network and to the Internet by means of several traffic concentrators. Particularly, we assume a set of N different access networks, where network i contains n^i HAS clients, streaming video at the same time. The problem we want to solve is twofold. First, clients have to obtain the highest possible video quality. Second, they have to show similar performance if they share bottleneck links, i.e., fairness. Based on this consideration, all the clients sharing the same bottlenecks should act fairly, even if they belong to different access networks.

The formal characterization of the fairness problem is given in Definition 1. We define K_j^i to be the number of segments the video content streamed by client j in network i is composed of, ${q_{max}}_j^i$ the highest available quality level, $q_j^i(k)$ the quality level requested for the $k-th$ segment and q_j^i the vector containing all the quality levels requested by client j. q^i is the matrix containing all the quality levels requested by the clients belonging to network i. $d_j^i(k, q^1, \ldots, q^i, \ldots, q^N, \beta^i)$ represents the download time of the $k-th$ segment for client j, while $b_j^i(k)$ denotes the video player buffer filling level of client j when the $k-th$ segment download starts. β^i is the vector containing the available bandwidth for the HAS clients belonging to network i, i.e., what remains of the network capacity once all the non-HAS traffic has been served.

Definition 1. *Fairness Optimization Problem*

$$\begin{aligned} \mathrm{J}(\mathrm{q}^{\mathrm{i}}) =& \xi \times \mathrm{QualityIndex}(\mathrm{q}^{\mathrm{i}}) - (1-\xi) \times \mathrm{FairIndex}(\mathrm{q}^{\mathrm{i}}) \\ & \text{with} \quad \xi \in [0;1],\ \mathrm{q}^{\mathrm{i}} = (\mathrm{q}_1^{\mathrm{i}}, \ldots, \mathrm{q}_{\mathrm{n}^{\mathrm{i}}}^{\mathrm{i}}) \end{aligned}$$

$$\begin{aligned} \mathrm{G}(\mathrm{J}(\mathrm{q}^1), \ldots, \mathrm{J}(\mathrm{q}^{\mathrm{N}})) =& \nu \times \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{J}(\mathrm{q}^{\mathrm{i}}) + \\ -(1-\nu) \times & \sqrt{\frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \left(\mathrm{J}(\mathrm{q}^{\mathrm{i}}) - \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{J}(\mathrm{q}^{\mathrm{i}}) \right)^2} \quad \text{with} \quad \nu \in [0;1] \end{aligned}$$

$$
\begin{aligned}
&\underset{q^1,\ldots,q^N}{\textit{maximize}} && \mathrm{G}(\mathrm{J}(\mathrm{q}^1),\ldots,\mathrm{J}(\mathrm{q}^\mathrm{N})) \\
&\textit{subject to} && 1 \leq q_j^i(k) \leq {q_{max}}_j^i \quad \forall i = 1\ldots N\ ,\forall j = 1\ldots n^i\ ,\forall k = 1\ldots K_j^i \\
& && d_j^i(k, q^1,\ldots,q^i,\ldots,q^N,\beta^i) \leq b_j^i(k) \\
& && \forall i = 1\ldots N\ ,\forall j = 1\ldots n^i\ ,\forall k = 1\ldots K_j^i
\end{aligned}
$$

The function $J(q^i)$ represents the performance of network i and is the linear combination of two terms. $QualityIndex(q^i)$, measures the overall video streaming quality of the clients in network i, while $FairIndex(q^i)$ measures fairness. By maximizing the linear combination of the average of $J(q^1),\ldots,J(q^N)$ and of the negative of their standard deviation, we aim to achieve the highest possible performance for every network while keeping the deviation among them as low as possible. Depending on applications and scenarios, ν can be tuned to benefit one of the two terms.

The final formulation of $QualityIndex$ and $FairIndex$ depends on the focus given to the multi-client optimization problem. From an application-aware point of view, the focus is on the user perceived video quality, denoted as QoE. In this case, $QualityIndex$ *can be characterized as the average of clients' QoE, while* $FairIndex$ *as the standard deviation of this average*. This way, we aim to maximize clients' QoE and keep the deviation as low as possible. The QoE model used in this chapter is presented in Section 3.4. From an application-agnostic point of view, the focus is on clients' bitrate. With this formulation, we are interested in fairness from a network point of view, i.e., clients have to fairly share the available bandwidth. Consequently, $QualityIndex$ *and* $FairIndex$ *can be computed as the average of clients' bitrate and as the standard deviation of this average*, respectively. In the design of our client, we focused on the application-aware interpretation, since it is directly correlated to the user perceived video quality.

In light of the above, it is clear why $QualityIndex$ and $FairIndex$ have to be optimized together. If we only optimize fairness, clients could obtain similar but unacceptable video qualities. Instead, our goal is also to reach the highest possible video quality. Depending on applications and scenarios, ξ can be tuned to benefit one of the two terms.

The second constraint of the optimization problem is intended to avoid freezes in the video playback. The download time of the next segment d_j^i has to be lower than the video player buffer filling level when the download starts, $b_j^i(k)$. This way, the video player buffer will never be empty and freezes are avoided. It is worth noting that the download time of the $k-th$ segment is a function of the quality levels downloaded simultaneously by all the other clients $(q^1,\ldots,q^i,\ldots,q^N)$ and of the available bandwidth β^i.

References

[1] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. *What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?* In 22nd International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV '12, pages 9–14. ACM, 2012.

[2] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. *Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale*. IEEE Journal on Selected Areas in Communications, pages 719–733, 2014.

[3] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck. *A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming*. In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–9, May 2014.

[4] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptive Video Players over HTTP*. Signal Processing: Image Communication, 27(4):271–287, 2012.

[5] C. Müller, S. Lederer, and C. Timmerer. *An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments*. In 4th Workshop on Mobile Video, MoVid '12, pages 37–42. ACM, 2012.

[6] C. Zhou, X. Zhang, L. Huo, and Z. Guo. *A control-theoretic approach to rate adaptation for dynamic HTTP streaming*. In 2012 IEEE Visual Communications and Image Processing Conference (VCIP), pages 1–6, 2012.

[7] G. Tian and Y. Liu. *Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming*. In 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 109–120. ACM, 2012.

[8] V. Adzic, H. Kalva, and B. Furht. *Optimized adaptive HTTP streaming for mobile devices*. 2011.

[9] D. Jarnikov and T. Ozcelebi. *Client intelligence for adaptive streaming solutions*. In 2010 IEEE International Conference on Multimedia and Expo (ICME), pages 1499–1504, 2010.

[10] S. Xiang, L. Cai, and J. Pan. *Adaptive Scalable Video Streaming in Wireless Networks*. In Third Annual ACM Conference on Multimedia Systems, MMSys '12, pages 167–172. ACM, 2012.

[11] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. *Design and Optimization of a (FA)Q-Learning-based HTTP Adaptive Streaming Client*. Connection Science, 26(01):27–45, 2014.

[12] B. Villa, P. Heegaard, and A. Instefjord. *Improving Fairness for Adaptive HTTP Video Streaming*. In R. Szabo and A. Vidacs, editors, Information and Communication Technologies, volume 7479 of *Lecture Notes in Computer Science*, pages 183–193. Springer Berlin Heidelberg, 2012.

[13] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive*. IEEE/ACM Transactions on Networking, 22(1):326–340, Feb 2014.

[14] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate Adaptation for Dynamic Adaptive Streaming over HTTP in Content Distribution Network*. Image Communication, 27(4):288–311, 2012.

[15] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. *ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)*. In 2013 International Packet Video Workshop (PV), pages 1–8, Dec 2013.

[16] C. Müller, S. Lederer, and C. Timmerer. *A proxy effect analysis and fair adaptation algorithm for multiple competing Dynamic Adaptive Streaming over HTTP clients*. In 2012 IEEE Visual Communications and Image Processing Conference (VCIP), pages 1–6, 2012.

[17] L. De Cicco, S. Mascolo, and V. Palmisano. *Feedback Control for Adaptive Live Video Streaming*. In Second Annual ACM Conference on Multimedia Systems, MMSys '11, pages 145–156. ACM, 2011.

[18] R. Kuschnig, I. Kofler, and H. Hellwagner. *An Evaluation of TCP-based Rate-control Algorithms for Adaptive Internet Streaming of H.264/SVC*. In First Annual ACM SIGMM Conference on Multimedia Systems, MMSys '10, pages 157–168. ACM, 2010.

[19] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming*. In 2012 International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM), pages 336–342, 2012.

[20] R. Houdaille and S. Gouache. *Shaping HTTP Adaptive Streams for a Better User Experience*. In Third Annual ACM Conference on Multimedia Systems, MMSys '12, pages 1–9. ACM, 2012.

[21] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming*. In 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[22] K. J. Ma and R. Bartos. *HTTP Live Streaming Bandwidth Management Using Intelligent Segment Selection*. In 2011 IEEE Global Telecommunications Conference (GLOBECOM 2011), pages 1–5, 2011.

[23] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. *QDASH: A QoE-aware DASH System*. In Third Annual ACM Conference on Multimedia Systems, MMSys '12, pages 11–22. ACM, 2012.

[24] A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. G. Steinbach. *Quality-of-experience driven adaptive HTTP media delivery*. In 2013 IEEE International Conference on Communications (ICC), pages 2480–2485, 2013.

[25] 3GPP. *3GPP System - Fixed Broadband Access Network Interworking (3GPP TS 23.139 version 11.3.0 Release 11)*. pages 1–90, 2013. Available from: http://www.3gpp.org/DynaReport/23139.htm.

[26] R. d. O. Schmidt, R. Sadre, A. Sperotto, and A. Pras. *Lightweight link dimensioning using sFlow sampling*. In 2013 International Conference on Network and Service Management (CNSM), pages 152–155, Oct 2013.

[27] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-based Bandwidth-lookup Service for Bitrate Planning*. ACM Transactions on Multimedia Computing, Communications and Applications, 8(3):24:1–24:19, August 2012.

[28] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. *Adaptation algorithm for adaptive streaming over HTTP*. In 2012 International Packet Video Workshop (PV), pages 173–178, May 2012.

[29] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for estimating QoE of video delivered using HTTP adaptive streaming*. In 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pages 1288–1293, May 2013.

[30] C. Claus and C. Boutilier. *The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems*. In Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI '98/IAAI '98, pages 746–752. American Association for Artificial Intelligence, 1998.

4

Software-Defined Network-Based Prioritization to Avoid Video Freezes in HTTP Adaptive Streaming

S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen and F. De Turck

Published in International Journal of Network Management, July 2016.

⋆ ⋆ ⋆

While the previous chapter focused on a system-wide property like fairness, the goal of Chapter 4 is to reduce the occurrence of video freezes, which are the main factor influencing users' Quality of Experience (QoE). To this aim, a network framework is presented, based on the Software-Defined Networking (SDN) principle and OpenFlow. An Openflow controller is in charge of prioritizing the delivery of HTTP Adaptive Streaming (HAS) segments, based on the network conditions and the HAS clients' status. Particularly, the HAS clients' status is obtained without any explicit clients-to-controller communication, and thus no extra signaling is introduced into the network. Moreover, this OpenFlow controller is transparent to the quality decision process of the clients, as it assists the delivery of the segments but it does not determine the actual quality to be requested. The OpenFlow framework is evaluated through emulation in several multi-client scenarios.

Results show that the proposed approach can reduce freeze time due to network congestion by more than 50% when compared to a baseline client-based heuristic not supported by prioritization, without impacting the performance of cross-traffic applications.

4.1 Introduction

Nowadays, video streaming applications are responsible for the largest portion of the Internet traffic. Particularly, HTTP Adaptive Streaming (HAS) protocols have become very popular and can therefore be considered as the de-facto standard for video streaming services over the Internet. Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming and MPEG Dynamic Adaptive Streaming over HTTP (DASH) are examples of available HAS technologies. In a HAS architecture, video content is stored on a server as segments of fixed duration at different quality levels. Each client can request the segment at the most appropriate quality level on the basis of the locally perceived bandwidth. In this way, video playback dynamically changes according to the available resources. Such dynamic adaptation results in a smooth video streaming experience. Nevertheless, several inefficiencies still have to be solved in order to further improve users' Quality of Experience (QoE). As reported by Akshabi et al. and Riiser et al., current rate adaptation heuristics perform quality selection sub-optimally, especially when a sudden bandwidth drop occurs [1, 2]. This leads to unnecessary quality switches and video playout interruptions, which negatively affect the final QoE of the users. Similar conclusions are drawn in the 2015 Conviva report on HAS [3]. The report reveals that almost 25% of the analyzed HAS sessions exhibit at least one video freeze. This problem is mainly due to the unmanaged nature of current HAS technologies. This entails that the clients are not aware of the real network conditions nor are they assisted in improving the delivered QoE. In this chapter, we investigate the aforementioned problems arising in HAS under volatile bandwidth conditions. Particularly, we present an OpenFlow-based controller in charge of collecting information from the network nodes. Based on the current network conditions and a network-based prediction of the HAS clients' status, the controller can decide to prioritize the delivery of particular HAS segments in order to avoid video freezes at the clients.

OpenFlow represents one of the most common Software-Defined Networking (SDN) protocols, a recently proposed innovative network architecture. In SDN, the data forwarding plane of the packets is decoupled from their control plane. This allows to exploit network functionalities in a flexible and real-time manner. Consequently, SDN represents the ideal solution to introduce innovative management solutions in multimedia delivery networks.

The main contributions of this chapter are threefold. First, we present an OpenFlow-based framework to help the clients avoiding video freezes caused by network congestion. Particularly, we assume that congestion occurs in the edge or

aggregation network, where alternative routing is unavailable, and handle it using a prioritized queue. The main element of this framework is a controller, which has the fundamental role of prioritizing the delivery of particular HAS segments in order to avoid video freezes. This decision is based on the HAS clients' status and on measurement data collected from the network nodes. Second, HAS clients' conditions are estimated at the controller side without any explicit clients-to-controller communication. Consequently, no extra signaling or overhead is introduced into the network. Third, detailed experimental results are presented to characterize the gain of our OpenFlow-based framework compared to state-of-the-art HAS heuristics. Particularly, we model two types of applications that, together with HAS, represent more than 80% of the traffic currently flowing through the Internet, namely web-browsing and progressive download (PD) video streaming [4, 5]. This allows us to evaluate the performance of the proposed solution in presence of realistic Internet cross-traffic. We also provide an analysis on the scalability of the proposed framework and show that it can control up to several thousands of video clients at the same time.

The remainder of this chapter is structured as follows. Section 4.2 reports related work on HAS optimization. Next, Section 4.3 details the proposed OpenFlow-based framework both from an architectural and algorithmic point of view. In Section 4.4, we evaluate our solution through emulation and show its benefits compared to current HAS heuristics. Section 4.5 presents the main conclusions.

4.2 Related Work

Akshabi et al. present an analysis of the performance and drawbacks of some commercially deployed HAS heuristics, such as Microsoft Smooth Streaming, Netflix and Adobe players [1]. They show that current rate adaptation heuristics perform quality selection sub-optimally. Particularly, these heuristics fail to adapt to rapid bandwidth changes. As a result, interruptions in the video playout and unnecessary quality switches occur. Similar conclusions are drawn by Müller et al. based on tests of different HAS implementations using real bandwidth traces collected on a mobile network [6]. They also point out that the Microsoft Smooth Streaming client is able to achieve the highest average bitrate as well as a low number of quality switches.

Many rate adaptation heuristics have been proposed to alleviate the problems highlighted in the previous paragraph [7]. Zhuo et al. present a control-theory-based HAS client where the buffer filling level of the client is controlled close to a specific threshold [8]. A similar approach is also studied by Tian et al. [9]. Wu et al. propose a novel media access control protocol for wireless regional area networks to guarantee the quality of service of Scalable Video Coding (SVC) multimedia streams [10]. Adzic et al. suggest to add additional information into the video segments to enhance the quality decision algorithm [11]. The client can

then decide to request a higher or lower segment depending on its bitrate and the its intrinsic video quality. Jarnikov et al. study an algorithm based on a Markov Decision Process (MDP), which requires offline training [12]. Xiang et al. use a similar approach for SVC streaming and adopt a Markov model to describe the variations of wireless channel conditions [13]. Also the work presented by Claeys et al. is based on an MDP and Reinforcement Learning theory [14]. In this case, the solution to the MDP is computed online by means of the Q-Learning algorithm, without any a priori knowledge. All these algorithms share a common drawback. Even though purely client-based heuristics simplify the design and implementation of the algorithms, such heuristics fail in case of sudden bandwidth drops [2]. This failure leads to video freezes and consequently low QoE. This issue is further worsened in live video streaming scenarios, where the playout buffer has to be reduced in order to minimize the camera-to-display delay.

In order to solve this issue, we adopt in this chapter an in-network approach, where intermediary nodes are placed in the network to collect information regarding the status of the clients. Consequently, the network has a comprehensive view of the clients' conditions and can help them achieving a high QoE [15]. The use of an OpenFlow controller to optimize the behavior of the HAS clients has been studied by Egilmez et al. [16]. They propose to dynamically re-route HAS traffic to avoid congested links. As traffic re-routing is only possible in the core Internet service provider network while congested links mainly arise in edge network, this approach is not able to fully optimize the behavior of the HAS clients. Several other works apply traffic-shaping techniques to limit the bandwidth assigned to each client and to drive them to request a target bitrate. Houdaille et al. introduce a bandwidth manager inside the home gateway to manage the flows belonging to the different clients [17]. Based on clients' characteristics and network conditions, the bandwidth manager determines the target bitrate for each client, in order to fairly share the available bandwidth. The bandwidth manager then applies traffic shaping techniques to limit the bandwidth of each client and drive it to request the target bitrate. A similar approach is investigated by Georgopoulos et al. [18]. A centralized OpenFlow controller is used to allocate the bandwidth for each streaming device, in order to obtain fairness from a QoE point of view. The authors present a model to correlate video bitrate with video quality, which is used by the controller in the bandwidth allocation process. Mok et al. introduce a measurement proxy between the clients and the video streaming server, which estimates the available bandwidth and decides the highest possible quality level the clients can download [19]. Based on this information and their buffer status, the clients decide the most appropriate quality level to request. El Essaili et al. propose a QoE optimizer for wireless networks that computes the optimal rate for the streaming clients, based on the wireless channel conditions [20]. This value is then used by a QoE proxy, in charge of intercepting and rewriting clients' requests

to match the requested quality level with the optimal rate. The principal disadvantage of the aforementioned algorithms is the active role of the in-network elements in the quality decision process. This aspect entails an alteration of the classical HAS principle, as the network de-facto decides which quality level the clients can download. Moreover, these solutions present robustness issues in case of fault or malfunctioning of the in-network elements.

Based on these considerations, we develop a purely network-based OpenFlow controller to optimize the QoE delivered to the clients. This approach presents three advantages. First, the in-network computation can be kept very simple and consequently not computationally demanding, since the quality decision algorithm still runs at the client. Second, the controller is transparent for both the HAS clients and the HAS server, as the controller only supports the delivery of the segments to avoid a freeze but does not have any active role in the quality decision process of the clients. Third, it is more robust in case of fault or malfunctioning of the network equipment, as the clients can continue to operate (at a sub-optimal level) without the in-network system. The concept of an OpenFlow controller to help the clients avoiding video freezes has been firstly introduced in our previous work [21]. The difference with respect to the work presented in this chapter is threefold. First, the OpenFlow controller proposed in our previous work collects client-related feedback by introducing explicit clients-to-controller communication. Particularly, this feedback is embedded as an additional HTTP header to the GET message sent by a client when requesting a new segment to download. This entails that malicious clients can transmit incorrect information to take advantage of the prioritization system. In this chapter instead, we design an algorithm located at the controller, which allows to estimate the clients' status without any explicit clients-to-controller communication. As the clients' status is automatically estimated by the controller, only the clients in risk of a video freeze are granted a prioritized delivery. Second, we extended the controller logic proposed by our previous work to decide whether a segment should be prioritized or not, in order to further improve its performance. Third, two types of applications have been implemented in order to emulate a realistic mix of the background Internet traffic, i.e., web-browsing and PD video streaming, based on the work by Akhtar et al. [5]. This allows us to investigate the performance of the OpenFlow-based framework in presence of realistic cross-traffic and to quantify the effect of prioritization both on HAS and background traffic.

4.3 Proposed OpenFlow-based Framework

In this section, we detail the implementation of the proposed OpenFlow-based framework introduced in the previous sections. The main component of this framework is an OpenFlow controller deciding which segments should be prioritized in

order to avoid interruptions in the video playout of the clients. As the controller can collect feedback from the network nodes and estimate the clients' status, it has a comprehensive view of the network and clients' conditions, and can therefore take the best decision to maximize the clients' QoE.

Another important aspect of our solution is that the clients are aware of the prioritization status of the downloaded segments. This information is used by the clients to adjust their quality selection process. If the clients are not aware of the prioritization status of the downloaded segment, two problems arise in their quality decision process. First, the bandwidth perceived by the clients in case of prioritization does not match the real available bandwidth. Second, a prioritized segment entails that the decision of the client was not optimal or that a sudden bandwidth drop has occurred. Consequently, the prioritization status is used by the clients as an additional feedback on the quality of their rate adaptation process or on the network conditions.

In the remainder of this section, we provide an architectural description of the proposed framework (Section 4.3.1) and detail the controller heuristic to enforce prioritization (Section 4.3.2).

4.3.1 Architectural Description

As previously introduced, the OpenFlow controller helps the clients avoiding freezes in case of scarce network resources, e.g., bandwidth drops, by introducing prioritization in the delivery of the video segments. Prioritization is enforced in the network by using an OpenFlow-enabled switch, the so-called *prioritization switch*, which is equipped with a best-effort and a prioritized queue. Based on controller decisions, the prioritization switch enqueues the clients' segments in one of these queues. As far as the prioritization switch positioning is concerned, the switch should be located before the main bottleneck of the network. Potential bottlenecks can be identified by analyzing the underlying network architecture or at runtime by monitoring link conditions (e.g., if the traffic exceeds a certain percentage of the link capacity, a prioritization switch can automatically become active). In this chapter, we focus on a scenario where the HAS clients share a common main bottleneck link, which is typically a link in the edge or aggregation network [22]. This entails that the behavior of all the clients sharing this common bottleneck can be optimized by a single prioritization switch located before the bottleneck itself.

An illustrative sequence diagram of the proposed framework is shown in Figure 4.1. The OpenFlow controller intervenes each time a client requests a new segment from the HAS server and decides whether the analyzed segment should be prioritized or not. Particularly, the prioritization switch forwards to the controller, via an OpenFlow rule, the IP packets flowing from the client to the server. The controller can analyze these packets and identify the HTTP GET requests issued

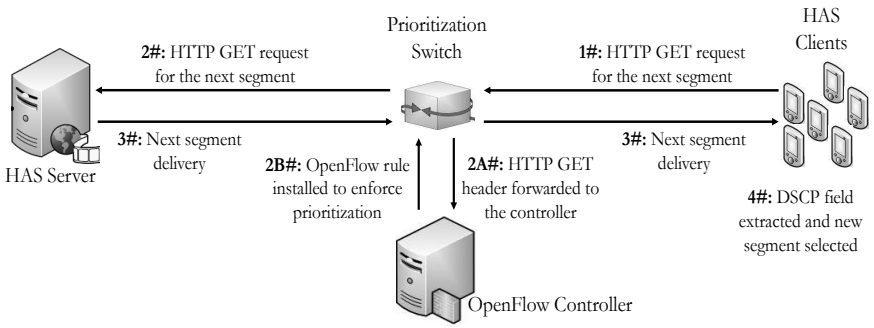


Figure 4.1: Logical sequence diagram of the proposed OpenFlow-based solution.

by the client. When a GET is received, the controller decides whether the delivery of the analyzed segment should be prioritized or not. This decision is based on network measurements collected from the prioritization switch and client related measurements, estimated at the controller side. Network measurements are obtained by using the OpenFlow protocol, which provides well-defined APIs to collect data from the OpenFlow switches. More specifically, the controller periodically polls the prioritization switch to compute the available bandwidth for the HAS clients in the best-effort queue (not shown in Figure 4.1). Client related measurements include the segment bitrate and the video player buffer filling level. The former is obtained by analyzing the GET message intercepted by the prioritization switch. The latter is estimated through buffer reconstruction methods [23]. This means that no explicit clients-to-controller communication is required in our framework. The prioritization logic implemented by the controller, as well as the buffer and the network bandwidth estimation algorithms, are presented in Section 4.3.2. Next, the controller installs a new OpenFlow rule on the prioritization switch to guarantee a proper delivery of the analyzed segment, i.e., best-effort or prioritized delivery. This way, the controller only supports the delivery of particular video segments, rather than determining the actual quality to be requested by the clients. This approach is robust toward controller and switch failures, as the clients can still operate even if prioritization cannot be enforced into the network. It is worth noting that the prioritization switch does not require any specialized feature to be used in the proposed framework. Particularly, the only required features are: (i) support the use of queues and (ii) support the OpenFlow protocol. Consequently, any general purpose OpenFlow-enabled switch can easily implement the prioritization switch functionalities.

As introduced previously, an important element of our solution consists of the prioritization-awareness of the HAS clients. Similarly to what is presented by Araujo et al., this communication is carried out by using in-network signaling instead of a direct communication channel between the controller and the clients

[24]. More specifically, the prioritization switch can be configured to mark prioritized packets with a specific Differentiated Services Code Point (DSCP) field. This field is extracted by the clients during the download of a segment to understand whether the segment was prioritized or not. As stated previously, this information is highly relevant for the quality decision process of the clients. When a client downloads a prioritized segment, the *prioritization mode* is triggered. In this mode, prioritized segments are ignored in the calculation of the estimated bandwidth because the bandwidth perceived in case of prioritization does not match the real network conditions. In addition, the client directly requests the next segment at the lowest quality. In this way, the client tries to minimize the risk of video freezes, which is high as the prioritization indicates. It is worth noting that the prioritization mode is independent from the actual rate adaptation heuristic implemented by the client. The only modification required at the client side is the extraction of the DSCP field from the downloaded segments and the prioritization mode, while no changes are required in the rate adaptation heuristic itself. This also entails that the proposed framework and the HAS clients can keep on operating even if the prioritization mode cannot be executed or is not implemented. Moreover, the proposed OpenFlow framework can be deployed to optimize the delivery of any existing heuristic. As the controller does not determine the actual quality to be requested by the clients but only supports the delivery of particular segments with a high risk of freezes, our solution can also be deployed in heterogeneous scenarios, where different clients are equipped with different rate adaptation heuristics. The proposed framework is also robust towards malicious clients that neglect the safety prioritization mode. The network controller knows when a particular client has been prioritized and is aware of the quality requested for the next segment by the client. When a client does not execute the prioritization mode, the controller can simply stop prioritizing it in the future. Another solution, which is more computationally expensive, would be to rewrite the HTTP GET of the client in order to force it requesting the lowest quality.

4.3.2 OpenFlow Controller

In this section we detail the operations performed by the controller to enforce prioritization in the delivery of HAS segments. The controller is designed to detect whether the download of a specific segment can lead to a video freeze. Particularly, we first present the operations performed by the controller to compute the available bandwidth for the HAS clients and the algorithm used to estimate the video player buffer filling level of the clients. Next, we detail the implementation of the prioritization logic embedded in the controller.

We assume in the following that all the clients are streaming from a single video server. Nevertheless, the proposed framework can be easily extended to

*Algorithm 1: HAS bandwidth estimation algorithm. The symbol * indicates the algorithm configuration parameters.*

Require: T_{poll}*, communication period between the controller and the prioritization switch, in seconds
α*, smoothing factor of the exponential average for the bandwidth estimation
queuesPort, port number of the prioritization switch where the best-effort and prioritized queues are installed
HasServerIP, IP address of the HAS Server
Ensure: *downstreamBwHas*, available bandwidth for HAS traffic, in Mbps

1: **while** $video\_sessions\_active$ **do**
2: $sleepSeconds(T_{poll})$
3: $downstreamBwHasTot =$ getTotalDownstreamBandwidthForHasTraffic$(HasServerIP)$
4: $downstreamBwHasPrio =$ getDownstreamBandwidthPrioritizationQueue$(queuesPort)$
5: $downstreamBwHasBe = downstreamBwHasTot - downstreamBwHasPrio$
6: $downstreamBwHas = \alpha \times downstreamBwHasBe + (1-\alpha) \times downstreamBwHas$
7: **end while**

support a multi-server scenario, where different clients stream videos from different servers at the same time. In this case, the controller should maintain a list of the IP addresses of the available video streaming servers. As an example, this list can be obtained off-line by the controller or on-line by analyzing the manifest files of the videos requested by the clients. No further modifications would be required to the algorithms presented in the following sections.

4.3.2.1 HAS Bandwidth Estimation

As introduced previously, the controller periodically polls the prioritization switch to compute an estimate of the available bandwidth for HAS traffic in the best-effort queue. This measurement is used in the prioritization logic to verify whether a normal best-effort delivery is sufficient to avoid a freeze in the video playout of the client. The communication between the controller and the switch is carried out using the OpenFlow protocol. Algorithm 1 reports the main steps performed by the controller.

Algorithm 1 is executed every T_{poll} seconds, with T_{poll} the communication period between the controller and the switch. The OpenFlow standard does not provide a direct API to collect the bandwidth information for a specific type of traffic (e.g., HAS traffic). Consequently, we adopt a two-steps approach to compute the available bandwidth for HAS traffic in the best-effort queue. The controller first obtains from the prioritization switch the total downstream bandwidth for HAS traffic *downstreamBwHasTot*, during the last T_{poll} seconds (line 3). This value represents the total amount of HAS traffic that has crossed the switch during the last measuring interval. This information is gathered by sending an OpenFlow aggregate flow statistic request message to the prioritization switch. This statistic

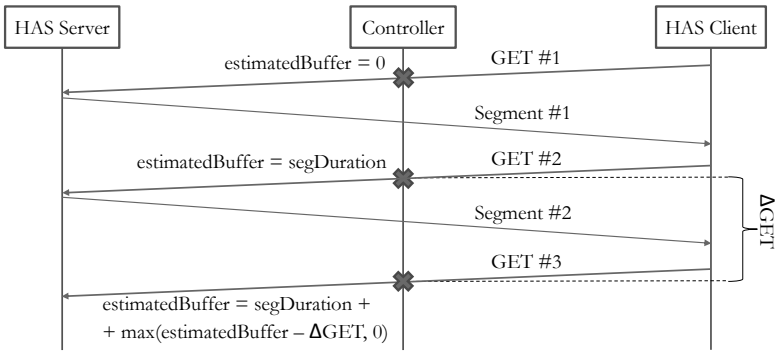


Figure 4.2: Logical sequence diagram illustrating the buffer estimation algorithm.

allows to obtain aggregate information about multiple flows with similar characteristics (e.g., all the flows with a specific source or destination IP address field). In order to obtain HAS-related measurements, the controller requests an aggregate statistic for all the flows whose IP source address matches that of the HAS Server $HasServerIP$. A flow belongs to the HAS group if the IP source address matches that of the HAS server. It is worth noting that *downstreamBwHasTot* represents the total downstream bandwidth for HAS traffic flowing through the prioritization switch, i.e., both in the prioritized and best-effort queue. In order to obtain an estimate of the available bandwidth for HAS traffic in the best-effort queue only, the controller also obtains the downstream bandwidth for HAS traffic in the prioritization queue, during the last T_{poll} seconds (line 4). We assume here that only the HAS clients are entitled to use the prioritized queue. This measurement is obtained by sending an OpenFlow queue statistic request message to the switch. The only argument of this request is the port number *queuesPort* of the prioritization switch where the queues are installed. Next, the bandwidth for the HAS clients in the best-effort queue is computed, as the difference between the total HAS downstream bandwidth and the prioritized bandwidth (line 5). The final throughput *downstreamBwHas* is calculated as the exponential average of the current throughput and past samples (line 6) and is used in the prioritization logic to decide whether a segment should be prioritized or not.

4.3.2.2 Buffer Estimation Algorithm

Together with network-related measurements, the controller calculates several parameters belonging to the HAS clients, namely the bitrate of the requested segments and the video player buffer filling level of the clients. It is worth stressing that these inputs are collected without any explicit feedback from the clients. The requested segment bitrate can be obtained by analyzing the GET message inter-

Algorithm 2: HAS buffer estimation algorithm. The algorithm is executed every time a GET request is intercepted.

Require: *segDuration*, duration of the video segments, in seconds
Ensure: *estimatedBuffer*, the estimated video player buffer filling level, in seconds

1: **if** First GET received from the client **then**
2: $estimatedBuffer = 0$
3: **else**
4: $\Delta GET = currentTime - previousGETTime$
5: $estimatedBuffer = \max(previousEstimatedBuffer - \Delta GET, 0) + segDuration$
6: **end if**
7: $previousGETTime = currentTime$
8: $previousEstimatedBuffer = estimatedBuffer$

cepted by the controller when the client requests a new segment. The buffer filling level can be estimated by analyzing the flow of GET messages sent by a client, as shown in Figure 4.2. When the controller intercepts a GET request for a new segment, it estimates the current buffer filling level of the client. We assume here the controller knows the duration of the video segments. As an example, this information can be obtained by analyzing the manifest file of the video downloaded by the clients. The possibility to intercept and analyze the video manifest has already been presented in the past, for example by Georgopoulos et al. [18], El Essaili et al. [20] and Li et al. [25]. When the first GET request from a client is intercepted, the estimated buffer level is set to zero, as the client has just started a new video session and its video player buffer is empty. Each time a new GET is intercepted, the controller is notified that the previously requested segment has been completely downloaded by the client and added to the video player buffer (see Figure 4.2). This condition always holds since a new GET is issued by a client only when the previously requested segment has been completely downloaded. The controller computes the time difference between the current GET and the previous one, which represents an estimate of the download time of the previously requested segment. The delta between the estimated download time and the segment duration indicates the amount of video playout time added to or subtracted from the client buffer. As an example, we assume the segment duration *segDuration* is 2 seconds, the time difference ΔGET between two subsequent GET is 5 seconds and the previously estimated buffer level *previousEstimatedBuffer* is 10 seconds. The new estimated buffer level can be computed as *previousEstimatedBuffer* - ΔGET + *segDuration*, i.e., 7 seconds.

A detailed description of the aforementioned operations is provided in Algorithm 2, which is executed each time the controller intercepts a GET request from an HAS client. When the controller receives the first GET from the client (line 1), the estimated buffer is assigned to zero (line 2), since the client has just started a new video session and its video buffer is empty. Otherwise, the controller com-

*Algorithm 3: Prioritization algorithm. The symbol * indicates the algorithm configuration parameters. The algorithm is executed every time a GET request is intercepted.*

Require: *prioBw*, bandwidth guaranteed to the prioritized queue, in Mbps
segDuration, duration of the video segments, in seconds
*safetyMarginDt**, safety margin in the estimation of the segment download time, in percentage
*criticalBufferPercentage**, critical percentage drop of the estimated buffer filling level
Ensure: *prioStatus*, prioritization decision. *false* means no prioritization

1: $downstreamBwHas = getAvailableBandwidthHasTraffic()$
2: $currentThrPrio = getCurrentThroughputPrioritizationQueue()$
3: $totHasClBe = getNumberHasClientsBestEffortQueue()$
4: $estimatedBuffer = getEstimatedBufferFillingLevel()$
5: $segBitRate = getSegmentBitRate()$
6: $segSize = segBitRate \times segDuration$
7: $estimatedDownloadTimeBe = (1 + safetyMarginDt) \times \frac{segSize}{downstreamBwHas/(totHasClBe+1)}$
8: $estimatedThrPrio = currentThrPrio + segBitRate$
9: **if** $estimatedDownloadTimeBe \leq estimatedBuffer$ **or** $estimatedThrPrio > prioBw$ **then**
10: $prioStatus = false$
11: $setTimerSafetyPrioritization(criticalBufferPercentage \times estimatedBuffer)$
12: **else**
13: $totHasClPrio = getNumberHasClientsPrioritizationQueue()$
14: $estimatedDownloadTimePrio = (1 + safetyMarginDt) \times \frac{segSize}{prioBw/(totHasClPrio+1)}$
15: **if** $estimatedDownloadTimePrio \leq estimatedBuffer$ **then**
16: $prioStatus = true$
17: **else**
18: $prioStatus = false$
19: **end if**
20: **end if**

putes the time delta between the current GET and the previous one (line 4). As mentioned above, this value indicates the amount of video played during the download of the previous segment. Next, the new estimated buffer is computed (line 5). The first addend *previousEstimatedBuffer* - ΔGET indicates the amount of time the video buffer has depleted during the download of the previous segment. The second addend indicates the amount of time the video player has increased at the end of the download, and is equal to the segment duration. The *max* operation avoids that the buffer filling level drops below zero if a video freeze occurs. Finally, the variables to be used at the next iteration are updated (lines 7-8).

4.3.2.3 Prioritization Logic

The OpenFlow controller helps the clients avoiding playout interruptions, by enforcing prioritization into the network. The controller logic is based on two types of inputs: the available bandwidth for HAS traffic in the best-effort queue

and an estimate of the buffer filling level of the client, computed in Algorithms 1 and 2, respectively. The decision on which segment to prioritize is carried out computing an estimate of the segment download time in the best-effort queue. If a best-effort delivery does not guarantee a timely download of the segment, i.e., if the download time is larger than the client buffer filling level, the segment is prioritized. Algorithm 3 details the operations performed by the controller.

As described in Section 4.3.1, Algorithm 3 is executed every time a client requests a new segment to download. First, the controller obtains the total available bandwidth for HAS traffic in the best-effort queue *downstreamBwHas* (computed in Algorithm 1) and the throughput of the prioritized queue *currentThrPrio* (lines 1-2). As explained in Section 4.3.2.1, this measurement can be easily obtained by using the OpenFlow protocol. The controller also obtains the number of HAS clients *totHasClBe* currently transmitting in the best-effort queue (line 3). Next, the estimated buffer filling level *estimatedBuffer* and the bitrate *segBitRate* of the requested segment are collected (lines 4-5), as presented in Algorithm 2. The size of the segment *segSize* is then computed as the product of the segment bitrate and the segment duration (line 6).

Next, the prioritization algorithm is executed to detect if current conditions can lead to a freeze (lines 7-20). The controller first computes an estimate of the segment download time in the best-effort queue *estimatedDownloadTimeBe*, as the ratio between the requested segment size *segSize* and the estimated bandwidth per-client in the best-effort queue (line 7). The former value is obtained by dividing *downstreamBwHas*, the available bandwidth for HAS traffic in the best-effort queue, by *totHasClBe*+1, the number of HAS clients currently transmitting in the best-effort queue plus one. We assume here the available bandwidth is shared fairly among the HAS clients. As *estimatedDownloadTimeBe* is only an estimate of the segment download time, the safety margin *safetyMarginDt* is introduced when computing it. As an example, given the estimated download time is 5 seconds and *safetyMarginDt* is 5%, *estimatedDownloadTimeBe* is computed as $5 + 5 * 0.05 = 5.25$ seconds. The algorithm proceeds as in the following. If *estimatedDownloadTimeBe* is smaller than the buffer filling level of the client (line 9), the segment is not prioritized because the risk of a video freeze is negligible. We assume here that the buffer decrease during the interval between the client's request and the GET analysis is negligible compared to the total buffer level. The controller also evaluates whether prioritizing the segment could congest the prioritized queue. Particularly, the current throughput of the prioritized queue plus the throughput of the analyzed segment (line 8) has to be lower than the guaranteed throughput *prioBw* (second condition on line 9). If this condition is not met, the segment is not prioritized. This way, the controller tries to avoid congesting the prioritized queue and thus negatively impacting the delivery of the other segments. Otherwise, the controller computes an estimate of the download time in the pri-

oritization queue *estimatedDownloadTimePrio* (line 14), similarly to what is done for the best-effort queue. Finally, the segment is entitled for prioritization only if *estimatedDownloadTimePrio* is lower than the estimated buffer level (line 15), i.e., only if a prioritized delivery is actually effective in avoiding a video freeze.

Even though the aforementioned algorithm is designed to detect conditions likely leading to a video freeze, it can fail prioritizing a segment because of two factors. First, the bandwidth per-client in the best-effort queue used to estimate the segment download time (line 7) is only an estimate of the real value. Second, the available bandwidth can drop *after* a decision has been taken, thus leading to a video freeze. For this reason, a safety prioritization condition is introduced, as shown on line 11. The controller sets a timer to be executed in $criticalBufferPercentage \times estimatedBuffer$ seconds. If a new GET is not received in this time interval, the previous decision is overridden and the segment is prioritized. The aforementioned timer is based on the estimated buffer filling level of the client and can be tuned through the parameter *criticalBufferPercentage*, which is expressed as a percentage of the estimated buffer size. If the controller detects that the buffer size has dropped by *criticalBufferPercentage* and that a new GET has not been received, then the current downloaded segment is prioritized as the video buffer is nearly depleted. This way, the controller tries to proactively react to a situation that could lead to a freeze. As an example, given *estimatedBuffer* and *criticalBufferPercentage* are equal to 7 seconds and 80% respectively, the safety prioritization timer is triggered after $7 \times 0.8 = 5.6$ seconds. It is worth noting that the prioritization mode at the client is triggered also in case of a safety prioritization.

As far as possible scalability issues are concerned, it is worth noting that the computation complexity of the controller heuristic is *O(n)*, with *n* the number of controlled clients. Thus, the controller heuristic scales well even for a large number of clients. Moreover, as explained in Section 4.3.1, a prioritization switch should be located before the main network bottleneck, which is typically a link in the edge or aggregation network. Consequently, only a few thousand clients would need to be managed by the controller at the same time in the worst case scenario. An analysis of the scalability of the proposed OpenFlow framework is presented in Section 4.4.5.

4.4 Performance Evaluation Results

4.4.1 Experimental Setup

The proposed OpenFlow framework has been implemented on the Mininet Network Emulator[1]. The HTTP server, where the video content is stored, is a Jetty

[1] http://mininet.org

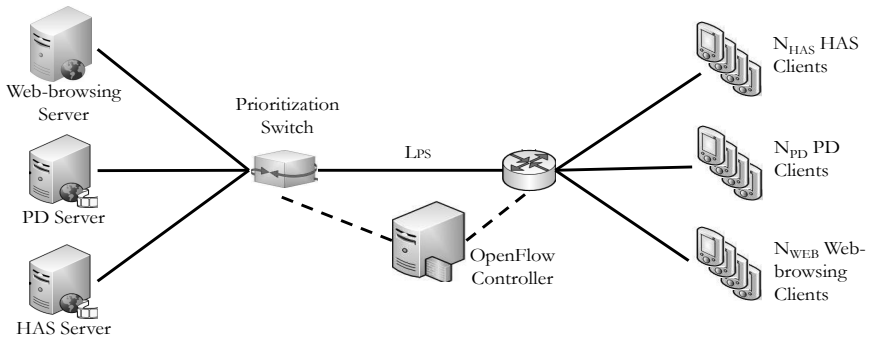


Figure 4.3: Emulated topology on Mininet.

Server [2]. The video streamed is *Big Buck Bunny*, composed of 299 segments, each 2 seconds long and encoded at 7 different quality levels: 300, 427, 608, 806, 1233, 1636, 2436 kbps. The HAS clients are implemented on top of the libdash library [26], the official reference software of the ISO/IEC MPEG-DASH standard. The Libpcap library[3] is used to extract the DSCP field from the received packets and thus enable prioritization-awareness. The rate adaptation heuristic embedded into the HAS clients is the FINEAS algorithm[4], presented in Chapter 3. The controller is implemented using POX[5], an extendible Python-based controller. Open vSwitch[6] 1.9.3 is used to realize the OpenFlow switches. The prioritization switch is equipped with a best-effort and a prioritized queue. A strict-priority policy was used for the experiments, i.e., the prioritized queue can transmit at a guaranteed rate of *prioBw* Mbps. The impact of rate *prioBw* on the performance of our solution is evaluated in Section 4.4.2. Based on our previous work [21], we set the switch polling time T_{poll} and the smoothing factor α in Algorithm 1 to 1 second and 0.25, respectively, and the estimated download time safety margin *safetyMarginDt* in Algorithm 3 to 5%. Preliminary simulations showed that 80% is the best choice for the *criticalBufferPercentage* parameter in Algorithm 3. For this reason, we kept it fixed for all the experiments.

The emulated network topology is shown in Figure 4.3, where the position of the prioritization switch is illustrated. The prioritized queue is installed on the interface towards link L_{PS}. N_{HAS} HAS clients stream the video sequence at the same time from the same HAS Server. In order to provide an extensive evaluation of the proposed framework, we emulate 30 episodes of the video trace and average the results over the 30 runs. The capacity on link L_{PS} is kept fixed during each

[2]http://eclipse.org/jetty
[3]http://www.tcpdump.org
[4]In this chapter, the in-network computation proposed in Chapter 3 has not been implemented.
[5]https://openflow.stanford.edu/display/ONL/POX+Wiki
[6]http://openvswitch.org

Table 4.1: Characteristics of the cross-traffic applications [5].

PD video streaming clients		**Web-browsing clients**	
Inter-request video time	*Video size*	*Inter-request page time*	*Page size*
Pareto distribution with mean 350 seconds and standard deviation to mean ratio of 2	Pareto distribution with mean 38 MBytes and standard deviation to mean ratio of 1.5	Pareto distribution with mean 8.5 seconds and standard deviation to mean ratio of 1.5	Based on web statistics [5]

Table 4.2: Cross-traffic loads used in the experiments (expressed as number of clients).

	HAS clients	**PD video streaming clients**	**Web-browsing clients**
Scenario A	30	30	15
Scenario B	30	21	9

episode while the capacity on all the other links is over-provisioned. During our evaluation, we tested different values of the fixed capacity of link L_{PS} in order to investigate the performance on the proposed solution under different levels of network congestion.

Cross-traffic for HAS clients is introduced by two other types of applications: N_{WB} web-browsing clients and N_{PD} PD video streaming clients. This applications mix allows us to evaluate the performance of the proposed framework under realistic network conditions and to analyze the impact of prioritization both on HAS and background traffic. The implementation of the cross-traffic applications used in this chapter is obtained by following the indications presented by Akhtar et al. [5]. PD clients can stream from a single PD server, shown in Figure 4.3. Web-browsing clients can download a web page from seven different web-servers, which has been shown to be the average number of web hosts used to download a web page [5] (for simplicity, only one host is shown in Figure 4.3). In order to download a web-page, a web-browsing client opens a single TCP connection to each of the available web-servers. Table 4.1 summarizes the most important characteristics of the cross-traffic applications. The inter-request video time and the size of the video to download for PD clients are both based on a Pareto distribution, as well as the inter-request page time for the web-browsing clients. The web page dimension is based on web statistics published by Google [5], and presents an average of 0.7 MBytes and a standard deviation of 1.5 MBytes. Two different traffic levels were used in our experiments, as shown in Table 4.2. In both scenarios, video streaming clients represent the majority of the total (74% and 77% of the clients in scenarios A and B, respectively). The percentage of video streaming traffic has been set according to the Sandvine report on Internet quality of experience, which shows that video streaming applications currently dominate Internet traffic [4]. As an illustrative example, Figure 4.4 reports the cross-traffic introduced by the PD and web-browsing clients for Scenario A and capacity on link L_{PS} fixed to 60 Mbps.

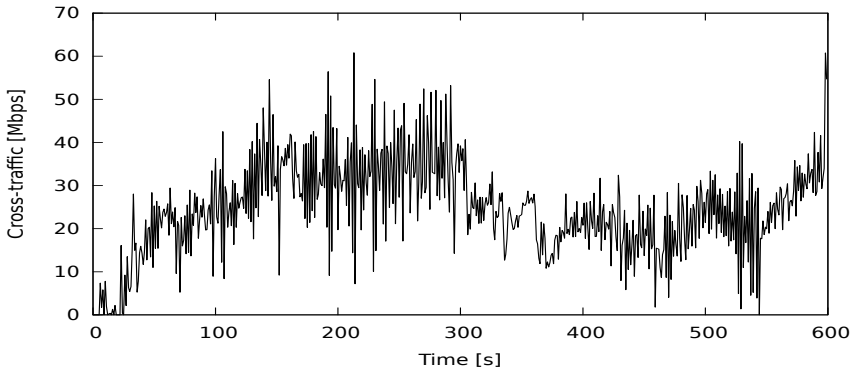


Figure 4.4: Example of the traffic introduced by the cross-traffic applications, for Scenario A. The capacity on link L_{PS} was fixed to 60 Mbps.

In order to provide an extensive benchmark of the proposed framework, we compare our results to those obtained using four other HAS clients. Particularly, we choose a popular proprietary HAS client, the Microsoft ISS Smooth Streaming (MSS) client [27], the FINEAS heuristic described in Chapter 3 without in-network prioritization, the purely buffer-based algorithm BBA client developed by Huang et al. [28] and the WiLo heuristic by Benno et al. [29]. The MSS heuristic selects the next quality based on the buffer filling level and the local perceived bandwidth. The client tries to control the buffer level between a lower and an upper threshold, while it requests the lowest available quality as soon as a panic threshold is crossed. The FINEAS heuristic is designed to maximize a utility function, which represents the QoE of the users. The client tries to maximize the requested quality, minimize the number of switches and control the buffer level close to a pre-defined threshold to avoid video freezes. The highest downloadable quality is constrained by the local perceived bandwidth, to avoid requesting qualities that could lead to a freeze in the near future. The BBA client is an example of a purely buffer-based algorithm, where the next quality is selected based on the buffer level only. Particularly, Huang et al. showed on a real field test that the BBA client is able to outperform the standard Netflix player. The WiLo client has been designed to handle the high bandwidth variability typical of wireless environments and to work with small video player buffer sizes. The video quality is increased only if the buffer level and the perceived bandwidth exceed certain thresholds for a specific amount of time. If one of the two conditions is not met, this timer is reset. The buffer level is also used to decide when to decrease the quality, based on a threshold. These thresholds can change over time to guarantee a timely rate adaptation. Unless otherwise stated, the buffer size for each client is equal to 5 segments, or 10 seconds.

Table 4.3: Overview of the evaluated network and traffic loads.

Parameter	Evaluated Values
L_{PS} *[Mbps]*	50, 60, 75
prioBw [Mbps]	$\{0\%, 5\%, 10\%, 15\%. 20\%\} \times L_{PS}$
Cross-traffic load	Scenario A, Scenario B

4.4.2 Evaluation of the Proposed Framework

In this section, a study is performed to investigate the best dimensioning for the proposed OpenFlow framework. Particularly, we aim to understand the impact that the amount of bandwidth reserved to the prioritized queue *prioBw* has on the performance of both the HAS clients and the cross-traffic applications, and select the configuration to use in the remainder of the chapter. In order to properly tune our HAS framework and select the best configuration, we evaluated the proposed OpenFlow framework under different network and traffic loads, as shown in Table 4.3. We varied the number of cross-traffic applications, the capacity assigned to link L_{PS} and the bandwidth assigned to the prioritized queue *prioBw*, for a total of 30 different configurations. The configuration where *prioBw* is assigned to 0% of the available capacity on link L_{PS} is used as reference behavior, as no prioritization can be enforced by the controller. As previously mentioned, the heuristic embedded into the HAS clients is the FINEAS heuristic presented in Chapter 3. In these experiments, all the clients are equipped with the same heuristic, in order to better understand the effects of prioritization. It is worth stressing that, as explained in Section 4.3.1, the proposed OpenFlow-based framework is not bounded to any specific rate adaptation heuristic, and can also be used to optimize the behavior of HAS clients equipped with different heuristics.

The results for these experiments are shown in Figure 4.5. Each graph reports on the x-axis the bandwidth percentage on link L_{PS} assigned to the prioritized queue, and on the y-axis the average freeze time per HAS client over the emulated episodes. For each emulated episode, we also computed the 10% and 90% quantiles, which quantify the maximum freeze time experienced by 10% and 90% of the HAS clients, respectively. Each point of the graphs is therefore associated with the average 10% and 90% quantiles over the emulated episodes. These values are used to understand the dispersion of the clients' freeze times around the average. In fact, the goal of the prioritization system is to reduce the average freeze time but also to assure that all the clients show similar performance. As expected, for both network load scenarios A and B, the average freeze time decreases as the channel capacity on link L_{PS} increases, even if prioritization is not enforced (e.g., *prioBw* equal to 0). As an example, the average freeze time for Scenario A drops from 3.69 seconds when L_{PS} = 50 Mbps (Figure 4.5a) to 0.45 seconds when L_{PS} = 75 Mbps (Figure 4.5e). When the channel capacity increases, less network conges-

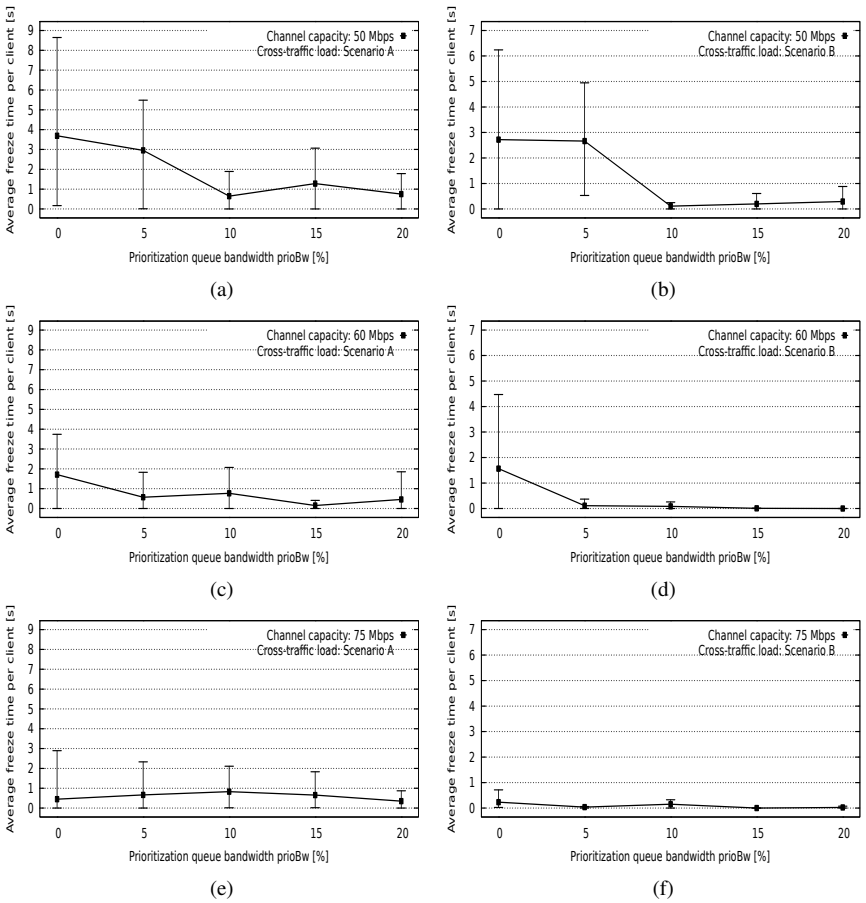


Figure 4.5: Analysis of the influence of the prioritized bandwidth prioBw on the average freeze time experience by the HAS clients, for different values of the channel capacity on link L_{PS}, for cross-traffic load as in Scenario A (4.5a, 4.5c, 4.5e) and B (4.5b, 4.5d, 4.5f).

tion occurs and, consequently, fewer freezes are experienced by the clients. The positive impact of prioritization on clients' performance is evident in the most congested scenarios (see Figures 4.5a-4.5b and 4.5c-4.5d). Convergence in the results is observed for a prioritized queue assigned to 10% or more of the total channel capacity, for both cross-traffic load scenarios. A prioritized queue limited to 5% of the channel capacity is able to reduce the 90% quantiles when L_{PS} is fixed to 50 Mbps, but not to consistently reduce the average freeze time. Only when L_{PS} is fixed to 75 Mbps (Figures 4.5e-4.5f), the effect of prioritization is negligible. As previously explained, this behavior is due to the minor network congestion arising

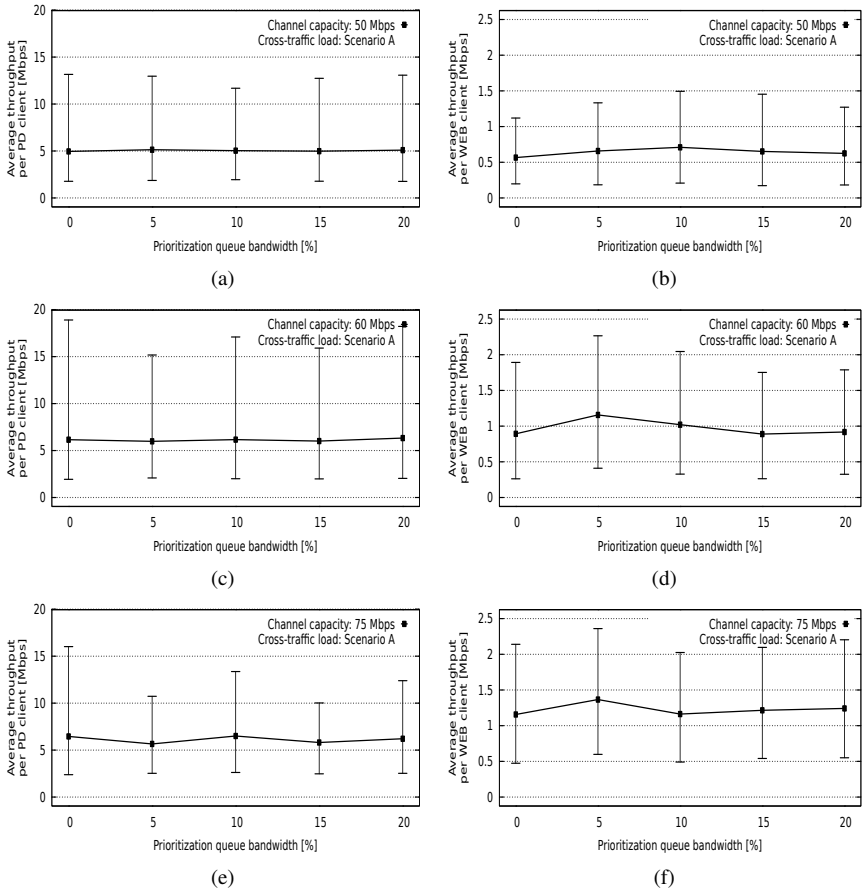


Figure 4.6: Analysis of the influence of the prioritized bandwidth prioBw on the average throughput obtained by the cross-traffic applications, for different values of the channel capacity on link L_{PS} for Scenario A, for PD clients (4.6a, 4.6c, 4.6e) and web-browsing clients (4.6b, 4.6d, 4.6f).

in this scenario. Nevertheless, the positive influence of prioritization can be noted in the 90% quantiles in Figure 4.5e, which decrease as prioritization is introduced into the network. This entails that the maximum freeze time experienced by the clients when prioritization is enforced is less than it would be without prioritization. This behavior is valuable as it guarantees that all the clients obtain similar performance.

In light of the proposed concept of net neutrality [30], an important question to be investigated is whether prioritization has any effect on the performance of the cross-traffic applications. Figure 4.6 reports the results of this analysis

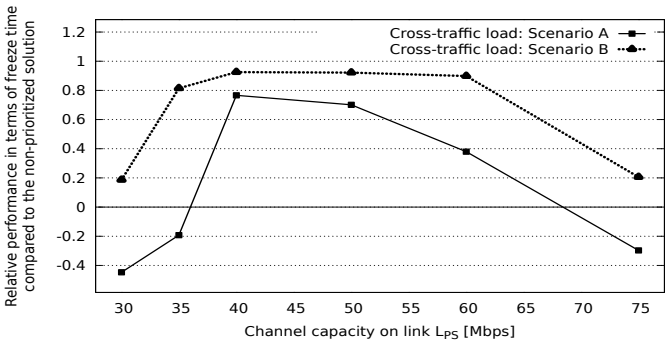


Figure 4.7: Relative performance of the proposed solution for several channel capacity conditions. The prioritized bandwidth is fixed to 10% of the capacity on link L_{PS}. Values above zero indicate a reduction of the average freeze time when prioritization is enforced.

for cross-traffic scenario A. Similar results are obtained for the cross-traffic scenario B, which are therefore omitted. Each graph reports on the x-axis the bandwidth assigned to the prioritized queue (in percentage), and on the y-axis the average throughput obtained by the PD and web-browsing clients over the emulated episodes. Each point of the graphs is associated with the average 10% and 90% quantiles over the emulated episodes. From the graphs, it appears that prioritization does not have any significant effect on the performance of the PD and web-browsing clients. Particularly, both the average throughput and the quantiles oscillate but no significant pattern can be detected. This behavior can be explained analyzing the amount of prioritizations enforced by the controller. Even in the worst case scenario in terms of network congestion (i.e., capacity on L_{PS} fixed to 50 Mbps and cross-traffic scenario A), the average number of prioritized segments per client is less than 10. This entails that, on average, less than 3.5% of the video is delivered in a prioritized way. As prioritization is not enforced massively in the network, the performance of the cross-traffic clients is not impacted. Moreover, as described in Section 4.3.1, when a segment is prioritized, the HAS client enters the *prioritization mode* and directly requests the lowest quality level. This approach helps reducing congestion occurring during bandwidth drops and frees resources that can be used by the cross-traffic clients to counter-balance the negative effect of prioritization. In light of the previous analysis on the performance of the HAS and cross-traffic clients, the bandwidth assigned to the prioritization queue has been finally fixed to 10% of the total channel capacity on link L_{PS}. This value is also going to be used in the remainder of the chapter.

In order to conclude the design study of the prioritization system, we present in Figure 4.7 an analysis of the performance of the prioritization system with a wider range of network congestion levels, obtained varying the capacity on link L_{PS} from

30 Mbps till 75 Mbps. The x-axis reports the capacity assigned to L_{PS}, while the y-axis reports the relative performance of the prioritized solution compared to the situation where no prioritization is enforced. Given F_{Prio} and F_{NoPrio} are the average freeze times with and without prioritization, each point of the graph is obtained as $(F_{NoPrio} - F_{Prio}) \,/\, (F_{NoPrio} + F_{Prio})$. Consequently, positive values indicate a reduction of the freeze time when prioritization is used. A clear trend can be identified in the performance of the proposed framework. Particularly, the relative gain of prioritization decreases when network congestion is very low (e.g., L_{PS} = 75 Mbps) or when it is extremely high (e.g., L_{PS} = 30 Mbps). As previously mentioned, when network congestion is low, prioritization is not mandatory, as the clients do not suffer from frequent and consistent bandwidth drops. In this case, both a prioritized and a non-prioritized delivery are able to almost completely eliminate video freezes, resulting in 0.82 and 0.44 second of freeze time on average, respectively. When congestion is instead extremely high, even a prioritized delivery is actually not able to consistently reduce the amount of freezes. In this case, too many clients should be prioritized at the same time because of the high level of congestion, while the controller is unable to guarantee this condition given the limited physical resources of the prioritized queue. Moreover, when the channel bandwidth is limited, the introduction of prioritization starts to have a negative effect on the HAS clients that are not prioritized. This means that prioritizing a client increases the risk of freezes for the others (due to the reduced available bandwidth), which in turn should be prioritized. This instability therefore reduces the performance of the proposed solution. Extremely high levels of congestion are generally due to a wrong planning of the network capacity, which is typically avoided in real deployments. Under normal network conditions (i.e., range 40-60 Mbps in Figure 4.7), the prioritization system is able to reach its optimal performance and almost completely eliminate video freezes for the HAS clients.

4.4.3 Rate Adaptation Heuristic Comparison

After the design study, we compare here the performance of our solution (from now on indicated with *Prio*) with that obtained using four other rate adaptation heuristics: the MSS, the WiLo, the BBA heuristic and a classical implementation of the FINEAS client (i.e., no prioritization is enforced into the network). We evaluate the performance of the different heuristics under different network and cross-traffic loads, as reported in Table 4.3 for the capacity on link L_{PS} and the cross-traffic scenarios. According to the design study performed in the previous section, the bandwidth *prioBw* assigned to the prioritized queue is fixed to 10% of the capacity on link L_{PS}. The performance metrics we will use to evaluate the performance of the different HAS clients are: (i) the client's freeze time, (ii) the requested video quality (from 1, the lowest, to 7, the highest), (iii) the stan-

dard deviation of the requested quality to measure the switching behavior of the clients. These three parameters are known to be amongst the most important factors influencing users' QoE in HTTP adaptive streaming [31]. All the results are reported using the averages over all the emulated episodes, together with the appropriate 10% and 90% quantiles. Figure 4.8 shows the obtained results, for both cross-traffic loads A and B. The x-axis of the graphs reports the channel capacity assigned to link L_{PS}, while the y-axis reports the average freeze time (Figures 4.8a-4.8b), the average requested quality (Figures 4.8c-4.8d) and the average quality standard deviation (Figures 4.8e-4.8f). In terms of freeze time (Figures 4.8a-4.8b), the BBA heuristic provides the worst performance, especially when network congestion is high (e.g., L_{PS} equal to 50 Mbps). This can be explained considering that this heuristic is purely buffer-based, i.e., it does not consider any information on the available bandwidth in the quality decision process. This type of heuristics performs well when the client's buffer is large, but they are not able to provide an optimal adaptation when the buffer size is rather small, as in our experiments. A large buffer simplifies the rate adaptation process, but it should be avoided because it causes a long delay when streaming live content and a big memory occupation on the device. The WiLo and the Prio client are the ones with the fewest freezes in all network scenarios, outperforming the MSS and FINEAS heuristics, which provide intermediary results. As an example, our solution is able to reduce freezes by a factor 10, 5 and 2 compared to the MSS, FINEAS and WiLo heuristics, when network congestion is high (i.e., L_{PS} fixed to 50 Mbps and cross-traffic load A). The low dispersion indicated by the 90% quantiles also shows that all the clients obtain similar performance and confirms the effectiveness of the proposed prioritization system.

Table 4.4 summarizes the results for the average freeze time. Each value reports the average freeze time together with the 10% and 90% quantiles, for the different heuristics and the different network scenarios. The proposed prioritization system is able to maintain the average freeze time below 1 second for all network and cross-traffic scenarios. Together with the freeze time, we also report the results for the average requested video quality (Figures 4.8c-4.8d). In order to maximize users' QoE, a heuristic should be able not only to avoid freezes, but also to request the highest possible video quality. In all network and cross-traffic scenarios, the BBA, FINEAS and our Prio solution are able to provide the best performance from this point of view. The increased quality has nevertheless a strong impact on the freeze time for the BBA heuristics, as shown in Figures 4.8a and 4.8b. On the contrary, the WiLo heuristic exhibits a very conservative behavior. As previously reported, this has a strong impact on the freeze time: downgrading the quality reduces the risk of freezes. In the cross-traffic scenario A and channel capacity fixed to 75 Mbps, for example, our Prio solution obtains a 30% higher quality compared to WiLo, with a similar amount of freezes. Figures 4.8e and 4.8f

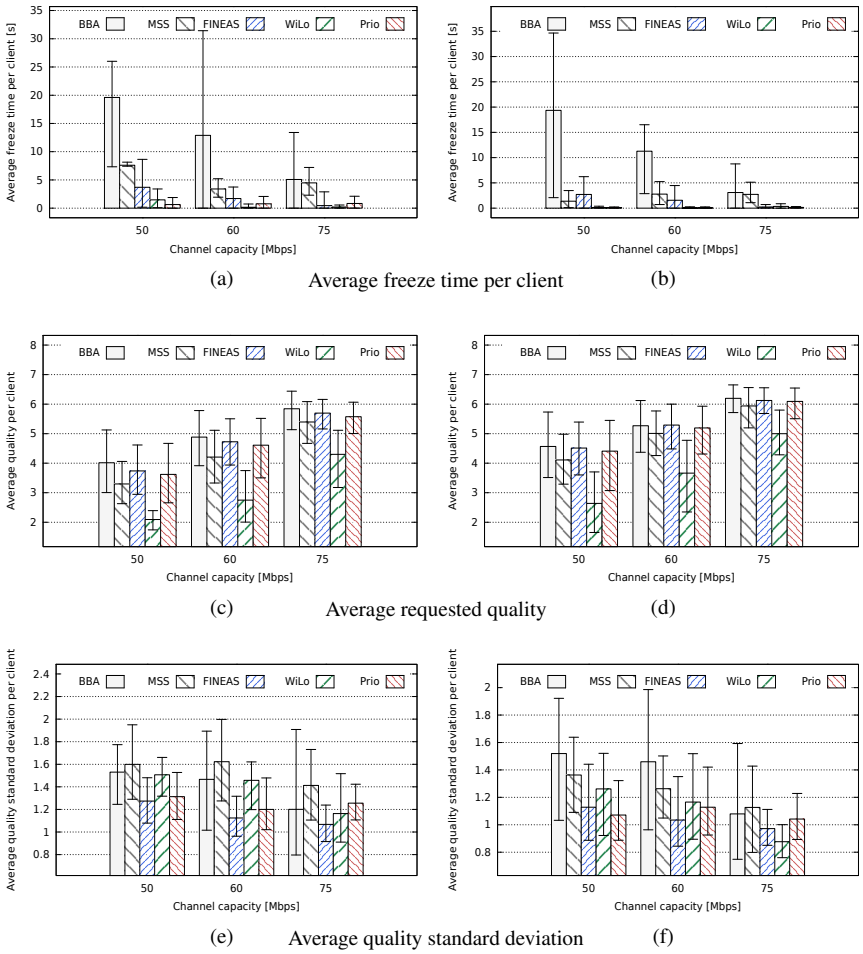
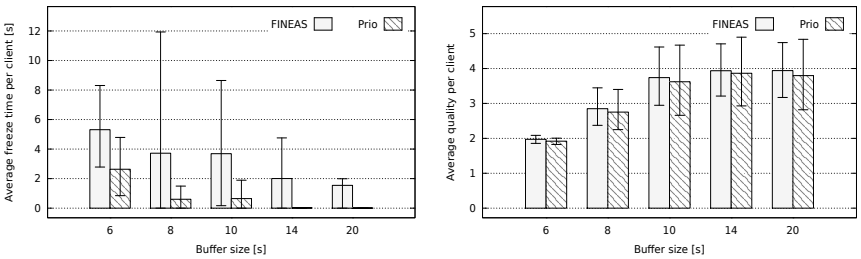


Figure 4.8: Comparison between the different clients for different values of the channel capacity on link L_{PS}, for cross-traffic load as in Scenario A (4.8a, 4.8c, 4.8e) and B (4.8b, 4.8d, 4.8f), in terms of average freeze time, requested quality and quality standard deviation.

depicts the results for the quality switches. In this case, the FINEAS heuristic is able to provide the best outcome. The slightly worse performance shown by our prioritization solution is mainly due to the prioritization mode, where a client is forced to request the lowest quality level in case the most recently downloaded segment has been prioritized. This entails the client is forced to switch every time a prioritized delivery is enforced.

Table 4.4: Performance summary for the different network scenarios in terms of freeze time. The average value is reported (in seconds), together with the 10% and 90% quantiles (in brackets).

	L_{PS} [Mbps]	**Prio**	**WiLo**	**FINEAS**	**MSS**	**BBA**
Scenario A	*50*	0.6(0-1.9)	1.4(0.1-3.4)	3.7(0.2-8.6)	7.6(7.4-8.1)	19.6(7.3-26.1)
	60	0.7(0-2.1)	0.2(0-0.7)	1.7(0-3.7)	3.4(1.9-5.2)	12.9(0-31.4)
	75	0.8(0.01-2.1)	0.2(0-0.6)	0.4(0-2.9)	4.5(2.3-7.2)	5.1(0-13.4)
Scenario B	*50*	0.1(0-0.2)	0.1(0-0.4)	2.7(0-6.2)	1.4(0.1-3.5)	19.4(2.1-34.6)
	60	0.08(0-0.3)	0.08(0-0.3)	1.6(0-4.4)	2.8(0.7-5.2)	11.3(2.9-16.5)
	75	0.1(0-0.3)	0.3(0-0.9)	0.2(0.02-0.7)	2.7(1.1-5.1)	3.1(0-8.7)



(a) Average freeze time (b) Average requested quality

Figure 4.9: Comparison between the FINEAS heuristic and the proposed prioritization solution for different buffer sizes, for cross-traffic load as in Scenario A and channel capacity on link L_{PS} fixed to 50 Mbps.

4.4.4 Influence of the Buffer Size

In this section, we investigate the clients' performance using different values of the buffer size, from 6 to 20 seconds. Based on the results of the previous section, we decided to evaluate our solution and the purely client-based FINEAS heuristic, which outperforms the MSS, the BBA and the WiLo client in terms of overall QoE (see Figure 4.8). Figure 4.9 shows the obtained results, for a capacity on link L_{PS} fixed to 50 Mbps and cross-traffic load as in Scenario A. Each graph reports on the x-axis the video player buffer size (in seconds), and on the y-axis the average freeze time and average requested video quality. Each point of the graphs is associated with the average 10% and 90% quantiles over the emulated episodes. From Figure 4.9a, it appears that the freeze time decreases as the buffer size increases, for both the classical FINEAS heuristic and our prioritized solution. As explained in Section 4.4.3, a large buffer simplifies the rate adaptation process, since it reduces the risk of freezes and thus the dependency on the current available

Table 4.5: Performance summary of the analysis delay, for an increasing number of clients in the system. The average value over the 10 runs is reported, together with the confidence interval at 95%.

Number of clients	**1**	**10**	**100**	**1000**	**5000**	**10000**
Average analysis delay [ms]	8.5±.15	8.2±.26	12.7±.08	18.8±.03	50.5±0.11	732.4±1.1

bandwidth. Despite that, a large buffer should be avoided because it causes a long delay when streaming live content, an important factor when optimizing users' QoE for live video. Introducing prioritization in the delivery of the segment helps to consistently reduce the buffer starvation, even in the 6 seconds case. Moreover, the proposed solution is able to completely eliminate freezes when the buffer size is larger than 14 seconds. Interestingly, similar results in terms of average freeze time are obtained with the FINEAS heuristic with a buffer size of 14 seconds and our prioritized solution with a buffer size of 6 seconds. This is again particularly beneficial in live video streaming scenarios, where the playout delay between the user and the live signal has to be minimized.

As far as the video quality is concerned, the larger the buffer size, the higher the requested quality, while convergence is reached for a buffer size of 10 seconds (Figure 4.9b). The average values are similar for the purely client-based FINEAS heuristic and our prioritized solution. This is expected, since the FINEAS heuristic is also used by the clients when prioritization is active. The only difference is represented by the 10% quantiles, which are lower in the prioritized case. This is mainly due to the prioritization mode executed by the clients, which force the clients to request the lowest quality in case prioritization takes place. This has an impact on the average requested quality, which is lower in the prioritized case compared to the non-prioritized one.

4.4.5 Scalability Analysis

As described in Section 4.3, the OpenFlow controller acts as a centralized node that decides whether a requested HAS segment should be prioritized or not. In this section, we investigate the scalability of the proposed framework, in terms of number of clients that can be controlled at the same time. Particularly, we identify the potential main bottleneck of our solution in the real-time analysis of the GET messages sent by the clients, performed by the prioritization switch and the controller. In steady-state conditions, each client requests a new segment to download every T seconds, with T the segment duration. This entails that the controller has to analyze a new GET and make a new decision every T/N_{HAS} seconds, given N_{HAS} is the number of HAS clients streaming at the same time. As an example, assuming T is 2 seconds and N_{HAS} is 1000, a new GET is issued every 2 ms, on

average. This load could congest the prioritization switch and the controller, thus introducing a non-negligible delay in the analysis of the GET messages that could negatively affect the performance of our solution.

In order to investigate this issue, we measure the *analysis delay* between a GET request issued by a client and the moment a new decision is taken by the controller (i.e., the segment should be prioritized or not), for an increasing number of clients, namely 1, 10, 100, 1000, 5000 and 10000. This analysis delay accounts for: (i) the time a GET travels from the client to the prioritization switch, (ii) the processing delay at the prioritization switch, (iii) the time a GET travels from the prioritization switch to the controller and (iv) the processing delay at the controller. The measured delay gives a clear indication of the capability of the proposed framework to scale in real network scenarios and should be as small as possible. Due to the severe technical difficulties of experimenting with such a high number of clients, both in emulation and in a real testbed, we decided to artificially reproduce the behavior of the group of clients. Particularly, instead of having N_{HAS} clients each issuing a new GET every T seconds, we allow a subset of n_{HAS} clients issuing a new GET every $T \times (N_{HAS}/n_{HAS})$ seconds. Practically, this means that each of the clients reproduces the behavior of N_{HAS}/n_{HAS} clients. During our experiment, we fixed the ratio N_{HAS}/n_{HAS} to 20. Moreover, the Scapy tool[7] is used by the clients to generate HAS-like GET messages. This means that in our experiments no video is actually streamed from the server, since we are merely interested in investigating the scalability characteristics of the proposed framework. The GET messages generated by the clients are analyzed by the controller as described in Section 4.4 and the analysis delay is computed. The same network setup as in Figure 4.3 has been used, without cross-traffic applications. All the links in the network are over-provisioned. We repeat the experiments 10 times and average the results over the 10 runs. During each run, the streaming of a 300 seconds video with segment duration of 2 seconds is simulated, as explained above.

Results for this analysis are reported in Table 4.5. The minimum analysis delay is obtained in the 1 client scenario and is about 9 ms, and it slightly increases up to 1000 clients. Even in the 5000 clients scenario, the analysis delay can be kept limited to about 50 ms. The action enforced by the controller on the prioritization switch has to be timely enough to guarantee a prioritized delivery of the segments in risk of a video freeze. This entails that the analysis delay has to be consistently lower than the segment download time. As prioritization is generally enforced only when the bandwidth availability is low, and consequently the download time of a segment is in the order of seconds rather than milliseconds, an analysis delay of 50 ms can be considered acceptable. As an example, we consider the video as in Section 4.4.1, with segment duration fixed to 2 seconds and lowest quality equal to 300 kbps. Even if the client experiences a bandwidth of 1 Mbps (which

[7] http://www.secdev.org/projects/scapy/

actually does not necessitate a prioritized delivery), the download time would be 0.6 seconds, i.e., more than 10 times the analysis delay obtained in the 5000 clients scenario.

It is worth stressing that the 1000 and 5000 clients scenarios represent the target use-cases for the proposed OpenFlow framework. As explained in Section 4.3.1, the prioritization switch should be located before the main network bottleneck, which is usually represented by a link in the edge or aggregation network. This entails that only a few thousand clients need to be managed by the prioritization switch and the controller at the same time. It is worth noting that these results have been obtained using the software switch Open vSwitch and the python-based controller POX. In a real deployment, high-end devices are typically used for both the switch and the controller, which would have a positive impact on the overall scalability of the proposed OpenFlow framework. Scalability issues appear in the 10000 clients case, where the average analysis delay is about 730 ms. The largest part of this delay is accumulated in the communication between the prioritization switch and the controller. In this case, the number of GET messages that should be processed by the switch and forwarded to the controller is so high that many are queued in the switch and consequently suffer from an increased delay.

4.5 Conclusions

In this chapter, we presented an OpenFlow-based framework to improve the QoE of HAS clients and reduce interruptions in the playout of the video clients. This was necessary as state-of-the-art rate adaptation heuristics can still suffer from non-negligible video playout freezes in case of sudden traffic increases that lead to network congestion. This objective is achieved by introducing prioritization in the delivery of HAS segments. An OpenFlow controller collects information on the overall network conditions and the HAS clients' status and decides whether a particular segment has to be prioritized or not in order to avoid a freeze at the client. The clients are also aware of the prioritization status of the downloaded segments in order to react properly to prioritization. Extensive emulation experiments using Mininet have validated the effectiveness of the proposed approach. Particularly, we have compared our OpenFlow-based framework with the proprietary Microsoft ISS Smooth Streaming client [27], the FINEAS heuristic, the BBA heuristic [28] and the WiLo client [29]. A mix of HAS, web-browsing and progressive download video streaming clients have been emulated in order to investigate the effect of prioritization under realistic network conditions. In the evaluated network scenarios, we were able to show that our OpenFlow framework results in a reduction of video freeze time up to 10 times, when compared to the benchmark algorithms.

Future research includes the study of buffer estimation methods to handle the presence of user interactivity (as pause or playout repositioning), which are typical

in video on demand (VOD) scenarios. We also propose to study more complex network scenarios, where a multitude of bottlenecks may simultaneously be present. In this case, we would need a system of prioritization switches, which exchange information with one or more controllers to decide which segment to prioritize. These controllers should then be able to coordinate with each other in order to guarantee an end-to-end prioritization and reach the final goal, i.e., avoid the occurrence of a freeze at the client side.

Acknowledgment

The research was performed partially within the iMinds V-FORCE project (under IWT grant agreement no. 130655). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

References

[1] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptive Video Players over HTTP*. Image Commun., 27(4):271–287, April 2012.

[2] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. *A Comparison of Quality Scheduling in Commercial Adaptive HTTP Streaming Solutions on a 3G Network*. In Proceedings of the 4th Workshop on Mobile Video, pages 25–30, NY, USA, 2012. ACM.

[3] CONVIVA. *2015 Viewer experience report*. http://www.conviva.com/vxr-home/vxr2015/.

[4] Sandvine. *Exposing the Technical and Commercial Factors Underlying Internet Quality of Experience*. https://www.sandvine.com/trends/global-internet-phenomena/.

[5] S. Akhtar, A. Francini, D. Robinson, and R. Sharpe. *Interaction of AQM schemes and adaptive streaming with Internet traffic on access networks*. In Global Communications Conference (GLOBECOM), 2014 IEEE, pages 1145–1151, Dec 2014. doi:10.1109/GLOCOM.2014.7036963.

[6] C. Müller, S. Lederer, and C. Timmerer. *An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments*. In Proceedings of the 4th Workshop on Mobile Video, MoVid '12. ACM, 2012.

[7] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. Communications Surveys Tutorials, IEEE, PP(99), 2014. doi:10.1109/COMST.2014.2360940.

[8] C. Zhou, X. Zhang, L. Huo, and Z. Guo. *A control-theoretic approach to rate adaptation for dynamic HTTP streaming*. In Visual Communications and Image Processing (VCIP), 2012 IEEE, pages 1–6, 2012.

[9] G. Tian and Y. Liu. *Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming*. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 109–120. ACM, 2012.

[10] K. Wu and W. Kuo. *Game-based Cross-layer Channel Allocation with SVC-encoded Multimedia Streams in Cognitive Radio Networks*. International Journal of Network Management, 22(5):397–417, September 2012. Available from: http://dx.doi.org/10.1002/nem.1798, doi:10.1002/nem.1798.

[11] V. Adzic, H. Kalva, and B. Furht. *Optimized adaptive HTTP streaming for mobile devices*. In Proceedings of SPIE, Applications of Digital Image Processing, 2011.

[12] D. Jarnikov and T. Ozcelebi. *Client intelligence for adaptive streaming solutions*. In Multimedia and Expo (ICME), 2010 IEEE International Conference on, pages 1499–1504, 2010.

[13] S. Xiang, L. Cai, and J. Pan. *Adaptive Scalable Video Streaming in Wireless Networks*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 167–172. ACM, 2012.

[14] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. *Design and Optimization of a (FA)Q-Learning-based HTTP Adaptive Streaming Client*. Connection Science, 26, 2014.

[15] F. Wamser, A. Blenk, M. Seufert, T. Zinner, W. Kellerer, and P. Tran-Gia. *Modelling and performance analysis of application-aware resource management*. International Journal of Network Management, 25(4):223–241, 2015. Available from: http://dx.doi.org/10.1002/nem.1894, doi:10.1002/nem.1894.

[16] H. Egilmez, S. Civanlar, and A. Tekalp. *An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks*. Multimedia, IEEE Transactions on, 15, April 2013. doi:10.1109/TMM.2012.2232645.

[17] R. Houdaille and S. Gouache. *Shaping HTTP Adaptive Streams for a Better User Experience*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 1–9. ACM, 2012.

[18] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming*. In Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[19] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. *QDASH: A QoE-aware DASH System*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 11–22. ACM, 2012.

[20] A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. G. Steinbach. *Quality-of-experience driven adaptive HTTP media delivery*. In ICC, pages 2480–2485, 2013.

[21] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *Network-based dynamic prioritization of HTTP adaptive streams to avoid video freezes*. In Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, pages 1242–1248, May 2015. doi:10.1109/INM.2015.7140475.

[22] J. Gettys and K. Nichols. *Bufferbloat: Dark Buffers in the Internet*. Commun. ACM, 55(1):57–65, January 2012. Available from: http://doi.acm.org/10.1145/2063176.2063196, doi:10.1145/2063176.2063196.

[23] R. Huysegems, B. De Vleeschauwer, K. De Schepper, C. Hawinkel, T. Wu, K. Laevens, and W. Van Leekwijck. *Session Reconstruction for HTTP Adaptive Streaming: Laying the Foundation for Network-based QoE Monitoring*. In Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service, pages 1–9, 2012.

[24] J. Araujo, R. Landa, R. Clegg, and G. Pavlou. *Software-defined network support for transport resilience*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–8, May 2014. doi:10.1109/NOMS.2014.6838243.

[25] Z. Li, M. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh. *Network friendly video distribution*. In 2012 Third International Conference on the Network of the Future (NOF), pages 1–8, Nov 2012. doi:10.1109/NOF.2012.6463984.

[26] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer. *Libdash - An open source software library for the MPEG-DASH standard*. In Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on, pages 1–2, July 2013. doi:10.1109/ICMEW.2013.6618220.

[27] A. Zambelli. *The Microsoft ISS Smooth Streaming (MSS) Client*. https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming.

[28] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. *A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service*. In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 187–198, New York, NY, USA, 2014. ACM.

[29] S. Benno, A. Beck, J. Esteban, L. Wu, and R. Miller. *WiLo: A Rate Determination Algorithm for HAS video in wireless networks and low-delay applications*. In Globecom Workshops (GC Wkshps), 2013 IEEE, pages 512–518, Dec 2013. doi:10.1109/GLOCOMW.2013.6825039.

[30] J. Krämer, L. Wiewiorra, and C. Weinhardt. *Net neutrality: A progress report*. Telecommunications Policy, 37(9):794 – 813, 2013. Available from: http://www.sciencedirect.com/science/article/pii/S0308596112001450, doi:http://dx.doi.org/10.1016/j.telpol.2012.08.005.

[31] R. Mok, E. Chan, and R. Chang. *Measuring the quality of experience of HTTP video streaming*. In Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on, pages 485–492, May 2011. doi:10.1109/INM.2011.5990550.

5

A Machine Learning-Based Framework for Preventing Video Freezes in HTTP Adaptive Streaming

S. Petrangeli, T. Wu, T. Wauters, R. Huysegems, T. Bostoen and F. De Turck.

Published in Journal of Network and Computer Applications, September 2017.

⋆ ⋆ ⋆

The previous chapter presented a Software-Defined Networking (SDN) framework that can successfully avoid video freezes at the clients. Despite its good performance, this approach requires the network controller to know the client configuration, in terms of buffer behavior and streamed video. Moreover, the proposed algorithms are hardcoded and require a certain level of hand-tuning to reach the best performance. For this reason, this chapter presents a complete redesign of the prioritization logic, based on Machine Learning (ML) algorithms. This way, the controller can autonomously learn when an HTTP Adaptive Streaming (HAS) segment needs to be prioritized. The ML engine is based on the Random Undersampling Boosting (RUSBoost) algorithm and fuzzy logic, which can detect when a client is close to a freeze and drive the network prioritization to avoid it. This

decision is based on measurements collected from the network nodes only, without any knowledge on the streamed video or on the client's characteristics. In this chapter, the design of the proposed ML-based framework is detailed and compared to other benchmarking HAS solutions, under various video streaming scenarios. Particularly, extensive experimentation shows that the proposed approach can reduce video freezes and freeze time with about 65% and 45% respectively, when compared to benchmarking algorithms.

5.1 Introduction

Video streaming currently represents one of the most important applications running over the Internet. Traditional video delivery techniques using the RTP/RTSP protocol suite and progressive download have now been replaced by HTTP Adaptive Streaming (HAS) protocols, which can be considered as the de-facto standard for video streaming services nowadays. In HAS, the video is stored on a server and is encoded at different quality levels. Each version of the same video is also temporally segmented. The HAS clients are equipped with a heuristic to dynamically select the most appropriate quality level to download, based on information as the locally perceived bandwidth and the video player buffer filling level. Despite the capability of HAS solutions to adapt to varying network conditions, current solutions are still suffering from interruptions of the video playout, also called video freezes [1]. The 2015 Conviva report shows that almost 25% of the analyzed HAS sessions are affected by at least one freeze [2]. Moreover, this problem becomes more prominent during live streaming sessions, where the video player buffer has to be reduced as much as possible in order to minimize the end-to-end delay. This inefficiency is extremely detrimental for the Quality of Experience (QoE) of the users and consequently for video streaming providers.

In order to reduce the occurrence of video freezes, we propose in this chapter a network-based approach, where intermediate network elements are designed to support the delivery of the video. The proposed network framework is built upon the Software-Defined Networking (SDN) principle, a recently proposed innovative network architecture [3]. In SDN, the control plane of the packets is managed by a network controller, while the data forwarding plane is the responsibility of the SDN switches. This decoupling allows to exploit network functionalities in a flexible and real-time manner, as the SDN controller can be easily programmed to control low-level network resources. In this chapter, we present an SDN controller in charge of prioritizing the delivery of particular video segments in order to avoid video freezes at the clients. The main element of the controller is a Machine Learning (ML) engine, based on the Random Undersampling Boosting (RUSBoost) algorithm [4] and fuzzy logic, which can identify when a client is close to a video freeze and decide whether the client's segments should be prioritized or not. The

RUSBoost algorithm is particularly suited for classification problems affected by class imbalance, as in the freeze prediction case. In fact, only a small percentage of a video is usually affected by freezes, since in normal conditions the HAS principle is able to achieve a continuous playout.

The main contributions of this chapter are threefold. First, we present a ML-based framework to help clients avoiding video freezes caused by network congestion. Particularly, we assume that congestion occurs in the edge or aggregation network, where alternative routing is unavailable, and handle it using a prioritized queue. An SDN controller is equipped with a ML engine, which allows the controller to predict when an HAS client will experience a video freeze and decide whether the currently downloaded segment should be prioritized or not. This decision is based on measurement data collected from the network nodes only, with no extra signaling required from the clients. Second, we design a freeze predictor based on the RUSBoost classification algorithm, which is embedded in the network controller. The freeze predictor is designed to detect conditions possibly leading to a freeze at the clients. The freeze predictor does not require any a priori knowledge on the characteristics of the video, as the segment duration or the bitrates of the different quality levels, nor on the client's configuration, in terms of maximum buffer size and start-up buffering time. Although this information can enhance the performance of the controller's logic as shown in Chapter 4, it is difficult to obtain in a real environment. By properly training the freeze predictor, it is still possible to reduce the number of video freezes experienced by the clients, without making any assumption on the streamed videos and the clients themselves. Third, detailed experimental results are presented to show the performance of the proposed approach under different congestion levels, network topologies and streamed videos. We also evaluate the performance of several classification algorithms for the freeze prediction task, and show that the RUSBoost algorithm is able to achieve the highest accuracy.

The remainder of this chapter is structured as follows. Section 5.2 reports related work on HAS optimization. Next, Section 5.3 details the proposed machine learning-based framework both from an architectural and algorithmic point of view. In Section 5.4, we evaluate our solution through emulation and show its benefits compared to current HAS heuristics. Section 5.5 presents the main conclusions.

5.2 Related Work

Many rate adaptation heuristics have been proposed to optimize the QoE of HAS clients [5]. Yin et al. present a control-theory-based HAS client based on the model predictive control approach [6]. In the work by Li et al., the download of a chunk is scheduled to obtain a continuous average data rate sent over the net-

work and to maintain the buffer level close to a given threshold [7]. This approach also helps reducing bitrate oscillations when multiple HAS clients compete for the same bandwidth. The same problem is also investigated by Jiang et al [8]. They study different design aspects that may lead to fairness improvements, including the quality selection interval, a stateful quality level selection and a bandwidth estimation using an harmonic mean instead of a normal one. Sun et al. improve the throughput prediction of HAS clients by developing a prediction model based on past video sessions, which is built offline in a node located in the streaming provider network [9]. This model is then used online in the rate adaptation heuristic of the video clients, which remain the sole responsible of the actual quality adaptation. Even though purely client-based heuristics simplify the design and implementation of the algorithms, such heuristics can fail in case of sudden and unexpected bandwidth drops. This failure leads to video freezes and consequently low QoE. This issue is further worsened in live streaming scenarios, where the playout buffer has to be reduced in order to minimize the camera-to-display delay.

In order to solve this issue, we adopt in this chapter an in-network approach, where intermediary nodes are placed in the network to collect information regarding the status of the clients. Consequently, the network has a comprehensive view of the clients' conditions and can help them achieving a high QoE. As an example of such a principle, Ivesic et al. show the benefit of an application-aware QoE-driven connection admission control for general multimedia services in LTE networks [10]. Ganjam et al. present a hierarchical centralized control system to optimize the delivery of HAS streams [11]. The root controller periodically creates a general model of the streaming system, which is then used by the children controllers (on a coarser time-scale) to take decisions on the best QoE strategy for the video clients. A similar approach is also employed by Mukerjee et al. to optimize the performance of live video streams [12]. The use of an SDN controller to optimize the behavior of HAS clients has been studied by Egilmez et al. [13]. They propose to dynamically re-route HAS traffic to avoid congested links. Uzakgider et al. investigate the performance of an SDN-controller equipped with a Q-Learning algorithm to re-route traffic for layered adaptive streaming [14]. As traffic re-routing is only possible in the core Internet service provider network while congested links mainly arise in the edge network [15], these approaches are not able to fully optimize the behavior of HAS clients. Several other papers apply traffic shaping techniques to limit the bandwidth assigned to each client and to drive them to request a target bitrate. Georgopoulos et al. propose a centralized SDN controller allocating bandwidth for each streaming device, in order to obtain fairness from a QoE point of view [16]. The authors present a model to correlate video bitrate with video quality, which is used by the controller in the bandwidth allocation process. The use of an SDN controller to obtain video quality fairness among different HAS clients is also investigated by Cofano et al. [17]. The

principal disadvantage of the aforementioned algorithms is the active role of the in-network elements in the quality decision process. This aspect entails an alteration of the classical HAS principle, as the network de-facto decides which quality level the clients can download. Moreover, these approaches are not designed to foresee the occurrence of video freezes and avoid them. In our solution instead, the quality level decision is completely left to the clients. The SDN controller supports the delivery of particular segments to avoid a freeze but does not have any active role in the quality decision process of the clients.

In this chapter, we propose a ML-based framework to foresee the occurrence of video freezes for HAS clients and reduce them using network-based prioritization. This approach presents two advantages. First, the in-network calculation can be kept very simple and consequently not computationally demanding, since the quality decision algorithm still runs at the client. Second, the controller is transparent for both the HAS clients and the HAS server, as the controller only supports the delivery of the segments to avoid a freeze but does not interfere in the quality decision process of the clients. Moreover, it is more robust in case of fault or malfunctioning of the network equipment, as the clients can continue to operate (at a sub-optimal level) without the in-network system. A network-based prioritization framework was already presented in Chapter 4, where we analyzed the prioritization system in terms of dimensioning of the prioritization queue, scalability issues and its performance compared with several purely client-based solutions. Even though the presented approach showed excellent results, it is affected by two main drawbacks. First, the controller's logic needs to know the characteristics of the streamed video, in terms of segment duration and bitrates of the different quality levels. Second, it requires an estimation of the buffer filling level of the clients' video player. This in turn entails that the controller has to know the initial buffering time of the client (i.e., the amount of video buffered before the playout can start). As no extra signaling is foreseen between the clients and the controller, this information can only be obtained by intercepting the manifest file requested by a client before starting the video. This aspect limits the general applicability of the proposed framework in a real environment. The main contribution of this chapter is therefore a complete re-design of the controller's logic using a ML-based approach. A freeze prediction algorithm located at the controller is able to detect beforehand when a client is going to freeze and drive the network prioritization. The freeze prediction algorithm is trained to correlate information as the network bandwidth and the timing of consecutive segment requests issued by a client to the occurrence of a freeze. Consequently, the prediction does not require any a priori assumption on the video characteristics or the buffer behavior of the clients.

A ML-based approach to off-line detect the occurrence of video freezes in HAS has also been proposed by Wu at al. [18]. A decision tree is trained to identify whether a completed video session is affected by video freezes. In this chapter

instead, the identification is performed on-line and on a segment basis, rather than off-line and on the basis of a completed video session. Our machine learning framework is designed to foresee future freezes and avoid them, in contrast to identifying freezes that have already occurred as investigated by Wu et al. [18].

5.3 Machine Learning-Based Framework

In this section, we detail the implementation of the proposed ML-based framework introduced in the previous sections. The main component of this framework is an SDN controller, deciding which segments should be prioritized in order to avoid interruptions in the video playout of the clients. This decision is driven by a ML module, the so-called *freeze predictor*, which is trained off-line and used on-line to drive the network prioritization. Each time a client requests a new segment, the predictor identifies whether the client could experience a freeze. The proposed network framework is implemented using OpenFlow, which currently represents one of the most common SDN protocols.

Another important aspect of our solution is that the clients are aware of the prioritization status of the downloaded segments. This information is used by the clients to adjust their quality selection process. If the clients are not aware of the prioritization status of the downloaded segment, a problem could arise in their quality decision process, as the bandwidth perceived by the clients in case of prioritization does not match the real available bandwidth. Moreover, a prioritized segment entails that the decision of the client was not optimal or that a sudden bandwidth drop has occurred. Consequently, the prioritization status is used by the clients as an additional feedback on the quality of their rate adaptation process or on the network conditions.

In the remainder of this section, we provide an architectural description of the proposed framework (Section 5.3.1) and detail the ML-based controller heuristic to enforce prioritization (Section 5.3.2).

5.3.1 Architectural Description

As previously introduced, the OpenFlow controller helps the clients avoiding freezes in case of scarce network resources, e.g., bandwidth drops, by introducing prioritization in the delivery of the video segments. Prioritization is enforced in the network by using an OpenFlow-enabled switch, the so-called *prioritization switch*, which is equipped with a best-effort and a prioritized queue. Based on the decisions of the controller, the prioritization switch enqueues the clients' segments in one of these queues. As far as the prioritization switch positioning is concerned, the switch should be located before the main bottleneck of the network, which is typically a link in the edge or aggregation network [15]. Potential bottlenecks can

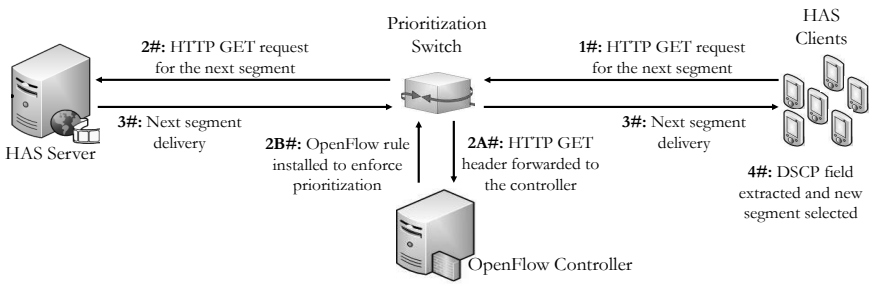


Figure 5.1: The OpenFlow controller intercepts clients' requests and decides whether the requested segment should be prioritized or not.

be identified by analyzing the underlying network architecture or at runtime by monitoring link conditions (e.g., if the traffic exceeds a certain percentage of the link capacity, a prioritization switch can automatically become active).

An illustrative sequence diagram of the proposed framework is shown in Figure 5.1. The OpenFlow controller intervenes each time a client requests a new segment from the HAS server and decides whether the analyzed segment should be prioritized or not. Particularly, the prioritization switch forwards to the controller, via an OpenFlow rule, the IP packets flowing from the client to the server. The controller can analyze these packets and identify the HTTP GET requests issued by the client. When a GET is received, the controller decides whether the delivery of the analyzed segment should be prioritized or not.

This decision is made by analyzing the flow of intercepted GET requests issued by a client and network measurements collected from the prioritization switch. Network measurements are obtained by using the OpenFlow protocol, which provides well-defined APIs to collect data from the OpenFlow switches. More specifically, the controller periodically polls the prioritization switch to compute the available bandwidth for the HAS clients in the best-effort queue (not shown in Figure 5.1). The freeze predictor module uses these inputs to detect whether the client is going to experience a freeze during the download of the requested segment. If prioritization is needed, a fuzzy module checks if enough resources are available in the prioritization queue to prioritize the segment and effectively prevent a freeze. The complete ML-based logic implemented by the controller, as well as the network bandwidth estimation algorithm, is presented in Section 5.3.2.

Next, the controller installs a new OpenFlow rule on the prioritization switch to guarantee a proper delivery of the analyzed segment, i.e., best-effort or prioritized delivery. This way, the controller only supports the delivery of particular video segments, rather than determining the quality to be requested by the clients. This approach is robust toward controller and switch failures, as the clients can still operate even if prioritization cannot be enforced into the network. It is worth noting

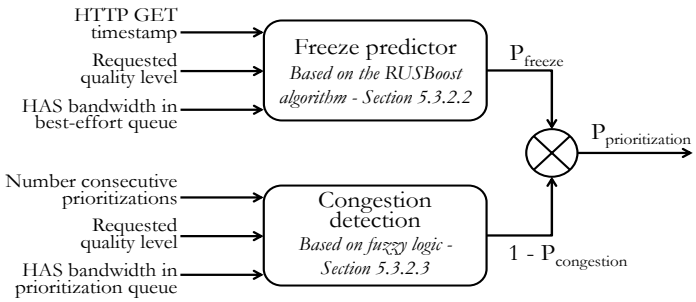


Figure 5.2: The controller's logic is based on a freeze predictor, to identify when a client is close to a freeze, and a congestion detection module, to check whether prioritizing the segment would congest the prioritization queue.

that the prioritization switch does not require any specialized feature to be used in the proposed framework. Particularly, the only required features for the switch are: (i) support the use of queues and (ii) support the OpenFlow protocol. Consequently, any general purpose OpenFlow-enabled switch can easily implement the prioritization switch functionalities.

As introduced previously, an important element of our approach consists of the prioritization-awareness of the HAS clients. This communication is carried out by using in-network signaling instead of a direct communication channel between the controller and the clients. More specifically, the prioritization switch can be configured to mark prioritized packets with a specific Differentiated Services Code Point (DSCP) field. This field is extracted by the clients during the download of a segment to understand whether the segment was prioritized or not. As stated previously, this information is highly relevant for the quality decision process of the clients. When a client downloads a prioritized segment, the *prioritization mode* is triggered. In this mode, prioritized segments are ignored in the calculation of the estimated bandwidth because the bandwidth perceived in case of prioritization does not match the real network conditions. In addition, the client directly requests the next segment at the lowest quality. This way, the client tries to minimize the risk of video freezes, which is high as the prioritization indicates. It is worth noting that the prioritization mode is independent from the actual rate adaptation heuristic implemented by the client. The only modification required at the client side is the extraction of the DSCP field from the downloaded segments and the prioritization mode, while no changes are required in the rate adaptation heuristic itself. This aspect also entails that the proposed framework and the HAS clients can keep on operating even if the prioritization mode cannot be executed or is not implemented.

5.3.2 OpenFlow Controller

The prioritization logic of the OpenFlow controller is executed each time a client requests a new segment to download, and is composed of two modules, the *freeze predictor* and the *congestion detection* modules. Figure 5.2 gives a high level overview of the interaction between these two modules in the prioritization decision of the controller. The freeze predictor module is designed to understand whether the client could experience a freeze during the download of the requested segment. The prediction is performed by analyzing the flow of GET requests issued by the client, the requested quality level and the available bandwidth for HAS traffic in the best-effort queue. The predictor is trained off-line using the RUSBoost classification algorithm. The congestion detection module is used to avoid congesting the prioritization queue and to allow a fair share among the different clients. This detection is performed by a fuzzy engine based on inputs as the bandwidth in the prioritization queue, the requested quality level and the number of consecutive prioritizations experienced by a client. The freeze prediction and the congestion detection modules return the probability for the current segment to freeze and the probability for the current segment to congest the prioritization queue, respectively. By combining these two factors, the controller obtains the actual probability of prioritizing the segment. A client is prioritized with higher probability when the risk of a freeze is high, as indicated by the freeze predictor, and prioritizing the segment does not congest the prioritized queue nor is unfair to the other clients, as indicated by the congestion detection module.

Section 5.3.2.1 details the operations performed to estimate the available bandwidth for HAS traffic in the best-effort queue. Sections 5.3.2.2 and 5.3.2.3 detail the freeze predictor and the congestion detection modules, respectively.

5.3.2.1 HAS Bandwidth Estimation

As previously introduced, the controller periodically polls the prioritization switch to estimate the available bandwidth for HAS traffic in the best-effort queue. We indicate with T_{poll} the polling period. The communication between the controller and the switch is carried out using the OpenFlow protocol. OpenFlow does not provide a direct API to collect bandwidth information for a specific type of traffic (e.g., HAS traffic). Consequently, we adopt a two-steps approach to compute the available bandwidth for HAS traffic in the best-effort queue.

The controller first obtains from the prioritization switch the total downstream bandwidth for HAS traffic during the last T_{poll} seconds. The controller sends an OpenFlow aggregate flow statistic request message for all the flows whose IP source address matches that of the HAS Server, which we assume is known to the controller. A flow belongs to the HAS group if the IP source address matches that of the HAS server. It is worth noting that this measurement represents the total

downstream bandwidth for HAS traffic flowing through the prioritization switch, i.e., both in the prioritized and best-effort queue. The controller also obtains the downstream bandwidth for HAS traffic in the prioritization queue during the last T_{poll} seconds. This measurement can be obtained using an OpenFlow queue statistic request message to the switch.

Next, the bandwidth for the HAS clients in the best-effort queue is computed, as the difference between the total HAS downstream bandwidth and the prioritized bandwidth. The final throughput is calculated as the exponential average of the current throughput and past samples. Based on the results of Chapter 4, we set the polling time T_{poll} and the weight of the exponential average to 1 second and 0.25, respectively.

5.3.2.2 Freeze Predictor Module

The freeze predictor is the core element of the proposed machine learning framework. When a segment is requested by a client, the freeze predictor has to decide whether the download of the segment is going to be affected by a freeze or not. This problem can be modelled as a classification problem with two classes: freeze and non-freeze. The classification is based on measurements obtained by the controller without any a priori assumption on the video and clients' configuration, in terms of video bitrates, segment duration, maximum buffer size and start-up buffering time. This aspect clearly complicates the classification task but allows a more general applicability of the proposed approach in a real environment. As the HAS clients do not explicitly communicate with the controller, the controller would need to intercept the video manifest to access the aforementioned information. Conversely, we only use data obtained by the OpenFlow protocol (e.g., the network bandwidth) or by analyzing the flow of HTTP GET requests issued by the HAS clients (e.g., the timing of the GET requests). In light of the above, the inputs for the freeze predictor are:

- ban_{HAS}, the bandwidth for HAS traffic in the best-effort queue when the segment is requested;
- Δban_{HAS}, the difference between two consecutive samples of the HAS bandwidth;
- ΔGET, the inter-arrival time between two consecutive GET requests issued by a client;
- $\overline{\Delta GET}$, the average GET inter-arrival time of a client;
- q, the requested segment quality level, which is expressed as an integer ranging from 0 (the lowest level) to q_{max} (the maximum quality level, different from video to video and not available to the controller);

- Δq, the difference between the qualities of two consecutive segment requests.

ban_{HAS} is measured using the OpenFlow protocol, as explained in Section 5.3.2.1. The risk of freezes is directly related to the network bandwidth: when congestion is high, more freezes are likely to occur. Δban_{HAS} indicates whether network conditions are improving or not. Intuitively, the risk of freezes decreases when network conditions improve, i.e., when Δban_{HAS} is greater than zero.

ΔGET is an important input for the freeze predictor, as it accounts for the buffer dynamic of the client. In steady-state conditions, a client issues a GET with a period equal to the segment duration. In this condition, no freezes can occur at the client. When the available bandwidth drops, the inter-arrival time ΔGET starts to increase, as the client has to wait for the complete download of a segment before requesting a new one. Consequently, the buffer starts to decrease and a possible freeze can occur. ΔGET allows to detect this condition and anticipate the freeze. The average GET inter-arrival time $\overline{\Delta GET}$ provides a rough estimation of the segment duration of the video, in steady-state conditions. This estimate allows to discriminate between videos with different segment durations and better exploit the information provided by ΔGET. As an example, a 4 seconds inter-arrival time is normal for a 4-seconds segment video, while it indicates that the buffer is depleting in a 1-second segment video. $\overline{\Delta GET}$ allows to discriminate between the two cases, as the average inter-arrival time is going to be different for different segment durations.

The last two inputs, q and Δq, account for the behavior of the client. A client usually requests a low quality when the bandwidth is scarce or the buffer is close to depletion. Conversely, a high quality is more susceptible to freezes if the bandwidth suddenly drops. This measurement is therefore highly valuable for the predictor. Δq shows if the client's conditions are improving. A client increases the quality only when the perceived bandwidth and/or the buffer allow it. The requested quality level can be extracted by analyzing the URL of the HTTP GET request. Different qualities must be associated with different URLs and this dissimilarity can be used by the controller to extract the requested quality.

By using these six inputs, the predictor is able to foresee the occurrence of a video freeze at the client. When a new segment is requested and a freeze is foreseen, the controller can enforce prioritization in order to avoid it. We divided this problem in two different phases: an off-line step where the predictor is built and an on-line step where the predictor is used to drive the network prioritization. The off-line phase is carried out collecting a large amount of HAS clients' logs in a controlled environment. Each entry of the training set is associated with a segment request made by a client, and is composed of the six aforementioned inputs and a label indicating whether the download of the requested segment resulted in a freeze or not. A machine learning algorithm can then be used to build the predictor.

Several challenges complicate the freeze prediction task. First, no assumptions are made on the clients' configuration and on the streamed videos. Second, the training set is imbalanced as most of the segment requests are not associated with a video freeze. In fact, the adaptive streaming principle is able to accommodate bandwidth variations, and network prioritization should only be used in emergency situations. Class imbalance is a well-known issue in the machine learning field, which can negatively affect the performance of classical classification algorithms [4]. In imbalanced training sets, the occurrence of the so-called negative class is much higher than the occurrence of the positive class, which represents the class of interest for the considered problem. In our case for example, most of the segments are not associated with a freeze, while we are interested in detecting segments affected by freezes (the positive class), which are the minority. Several algorithms have been proposed in literature to address this problem, with the RUSBoost algorithm emerging for its simplicity, accuracy and low computational complexity [4]. The RUSBoost algorithm combines two techniques: random undersampling and boosting. In random undersampling, most of the examples belonging to the negative class are removed from the training set, in order to obtain a desired balanced class distribution (e.g., 50%). This approach reduces the training time (as the new training set is much smaller than the original one), at the cost of a loss of information, as many negative class entries are removed. This drawback is counterbalanced using boosting. In boosting, a set of classifiers is iteratively built. At each iteration, the new classifier focuses on training examples that were misclassified at the previous iteration. After training, each classifier in the set, also called ensemble, participates in a vote to classify new examples. In the RUSBoost case, at each iteration, the new classifier is trained with a different undersampled subset of the original training set. Using this process, the loss of information due to undersampling is mitigated because data excluded in a certain iteration can be included in other ones.

In light of the above, the RUSBoost algorithm represents a viable choice for the freeze predictor task. As shown in Figure 5.2, the outcome of the freeze predictor is P_{Freeze}, the probability for the current requested segment to freeze. The classifiers in the ensemble are associated with a weight, indicating how good or bad a given classifier is. When classifying a new request, each classifier contributes to the final decision according to its weight. All the weights of the classifiers indicating that the current segment will not freeze are summed in the $W_{NoFreeze}$ variable. Conversely, all the weights of the classifiers indicating a freeze are summed in the W_{Freeze} variable. The output probability P_{Freeze} is simply computed as $W_{Freeze}/(W_{Freeze}+W_{NoFreeze})$.

In Section 5.4.2, the off-line training phase of the freeze predictor using the RUSBoost algorithm is analyzed in detail and compared with other state-of-the-art classification algorithms.

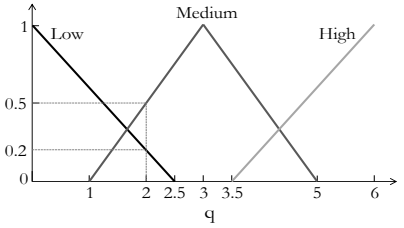
5.3.2.3 Congestion Detection Module

The second module completing the ML-based controller logic is the congestion detection module. This module is designed to understand whether prioritizing a particular segment would congest the prioritization queue. Prioritizing a segment is only useful when a freeze is probable and enough resources are available in the prioritization queue to effectively prevent the freeze. If too many clients are prioritized at the same time, the performance of the whole framework would drop. Moreover, this module ensures that the prioritization queue is fairly shared among all HAS clients, by limiting the maximum amount of consecutive prioritizations a client can benefit from. For each segment requested by a client, three inputs are used by the congestion detection module: (i) q, the segment's quality level, (ii) N_{Prio}, the number of consecutive prioritizations enjoyed by the client and (iii) $clientBan_{Prio}$, the bandwidth per client in the prioritization queue, computed as the ratio between the bandwidth in the prioritization queue and the number of prioritized clients. It is worth mentioning that $clientBan_{Prio}$ is only an estimate, as bandwidth is not always shared fairly among clients.

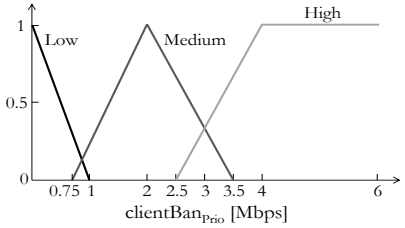
q and $clientBan_{Prio}$ allow to assess the risk of prioritizing the requested segment. Even though the bitrates of the video are not known to the controller, a higher quality segment will always take more resources to be transported than a lower quality segment. Consequently, the controller has to find a trade-off between prioritizing few high quality segments or many low quality segments. $clientBan_{Prio}$ is the fundamental input to understand the conditions of the prioritization queue. When many clients are prioritized at the same time, the bandwidth per client decreases and, consequently, also the probability of a successful prioritization. N_{Prio} is used to limit an unfair usage of the prioritization system. The probability of prioritization diminishes as the number of consecutive prioritizations increases, as to allow all clients to benefit from the system.

As explained in the previous paragraph, the correlation and the influence of the different inputs are rather simple. Despite that, an immediate translation into mathematical formulation is not straightforward. For this reason, we decided to implement the congestion detection module using fuzzy logic. Fuzzy logic allows to take complex decisions based on uncertain inputs, mimicking the reasoning of the human logic. It is based on fuzzy sets, which provide a rough modelling of the main characteristics of the analyzed problem, and fuzzy rules, which combine the knowledge modelled by the sets and make a final decision. These rules are expressed using human concepts rather than strict measurements. The fuzzy sets and the fuzzy rules are designed considering common sense knowledge of the analyzed domain. The fuzzy-based congestion detection module is composed of three input sets (one for each input), one output set for the output variable $P_{congestion}$ and six fuzzy rules to correlate the sets, which are described in the following.

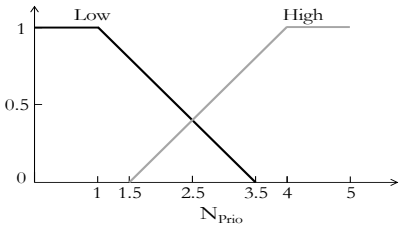
The fuzzy set associated to the quality q is composed of three membership



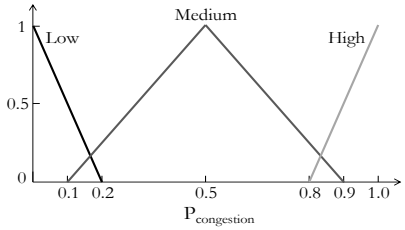
(a) Input q is divided into three symmetric input sets: low, medium and high.



(b) $clientBan_{Prio}$ is considered low when below 1 Mbps and high when above 2.5 Mbps.



(c) Only two membership functions compose the fuzzy set for N_{Prio}: low and high.



(d) $P_{congestion}$ is considered high when higher than 0.8 and low when below 0.2.

Figure 5.3: The fuzzy-based congestion detection module is composed of four fuzzy sets: three for each input (5.3a, 5.3b, 5.3c) and one for the output (5.3d).

functions: low, medium and high (Figure 5.3a). At any given point of the input range, the value of q is associated to one or more of these membership functions, with a certain degree. As an example, a value of q equal to 2 is considered 0% high, 50% medium and 20% low. On the contrary, a value of q equal to 0 (i.e., the lowest one) is considered 100% low.

Also $clientBan_{Prio}$ is composed of a low, medium and high membership function (Figure 5.3b). When the bandwidth per client drops below 1 Mbps, it starts to be considered low. Intuitively, the lower qualities of a video are encoded at low bitrates, usually below the 1 Mbps threshold. Consequently, if we want to assure a successful prioritization for at least the lower qualities of a video, a consistent amount of bandwidth should always be available in the prioritization queue. A bandwidth higher than 2.5 Mbps starts to be considered high, since even the higher quality segments (usually encoded at high bitrates) can be prioritized without congesting the prioritized queue.

Only two membership functions compose the fuzzy set for N_{Prio} (Figure 5.3c). In this case, a specific value of N_{Prio} is either considered low or high. A fine-grained representation of this variable is not needed in this case: only when a large number of consecutive prioritizations occur (e.g., higher than 3) the congestion detection module should avoid prioritizing the client.

Table 5.1: The six fuzzy rules of the congestion detection module.

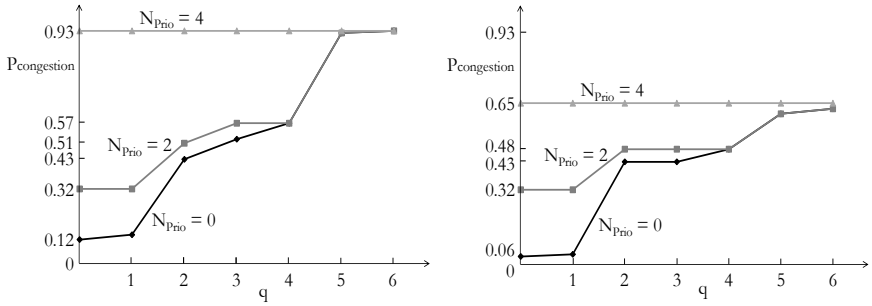
1. If $clientBan_{PRio}$ is HIGH then $P_{congestion}$ is LOW
2. If $clientBan_{PRio}$ is LOW then $P_{congestion}$ is HIGH
3. If N_{PRio} is HIGH then $P_{congestion}$ is HIGH
4. If q is HIGH and $clientBan_{PRio}$ is NOT HIGH then $P_{congestion}$ is HIGH
5. If q is LOW and $clientBan_{PRio}$ is NOT LOW then $P_{congestion}$ is LOW
6. If q is MEDIUM and $clientBan_{PRio}$ is MEDIUM and N_{PRio} is LOW then $P_{congestion}$ is MEDIUM

The output set for $P_{congestion}$ is also divided in low, medium and high (Figure 5.3d). When $P_{congestion}$ is higher than 80%, the probability of congesting the queue is considered high. Conversely, it is low when below 20% and considered medium otherwise.

Although fixed in this chapter, the fuzzy membership values can be easily modified by the service provider in order to reflect the specific conditions of the video streaming service, without altering the general design of the proposed fuzzy system. Another possibility would be to dynamically alter the fuzzy membership values using an additional learning loop [19].

The core of the fuzzy engine is represented by the fuzzy rules, which correlate the input and output sets and allow to make a decision (also called inference). The fuzzy rules for the congestion detection module are presented in Table 5.1. The first rule states that when the bandwidth is very high the probability of congesting the queue is considered low, independently from q and N_{PRio}. The opposite conclusion is drawn by the second rule. The third rule is designed to ensure a fair share of the prioritized system among all clients, by limiting the number of consecutive prioritizations for a client. The other rules take into account the requested quality level. As an example, a medium quality level will cause a medium congestion when the prioritization queue is in decent conditions and the client has not been prioritized too many times (rule six in Table 5.1).

Despite the simplicity of the proposed fuzzy model, a complex behavior can be obtained during the decision phase. As an example, Figures 5.4a and 5.4b show the decision plane of the fuzzy engine when fixing N_{PRio} to 0, 2 and 4 and $clientBan_{PRio}$ to 1 Mbps and 3 Mbps, respectively. The x-axis reports the requested quality level, while the y-axis the value of $P_{congestion}$. As expected, an increase of the bandwidth per client (i.e., less clients prioritized at the same time) results in a smaller probability of congestion. The highest value when $clientBan_{PRio}$ is 1 Mbps is 0.93 (Figure 5.4a), while this value drops to 0.65 when $clientBan_{PRio}$ is 3 Mbps (Figure 5.4b). $P_{congestion}$ follows an increasing monotonic trend with respect to the requested quality. In this case, the shape of the curves changes depending on the



(a) $clientBan_{Prio}$ = 1 Mbps (b) $clientBan_{Prio}$ = 3 Mbps

Figure 5.4: An example of the fuzzy decision plane obtained when $clientBan_{Prio}$ is fixed to 1 Mbps (5.4a) and 3 Mbps (5.4b). $P_{congestion}$ increases with the requested quality and the number of consecutive prioritizations, and decreases as the bandwidth per client increases.

Table 5.2: Characteristics of the HAS videos. The nominal average bitrate for the 2-seconds segment version is reported, together with the standard deviation (between brackets). All values are expressed in kbps.

Big Buck Bunny	*Of Forest and Man*	*Tears of Steel*	*Elephant's Dream*	*Red Bull Playstreets*
334(±66)	366(±75)	254(±105)	382(±123)	300(±41)
522(±75)	553(±101)	507(±217)	795(±176)	896(±134)
791(±101)	824(±168)	811(±346)	1494(±526)	1180(±226)
1244(±257)	1519(±376)	1516(±680)	2444(±728)	1993(±291)
1546(±417)	2529(±839)	2427(±1057)	3431(±1481)	2995(±376)
2494(±531)	3798(±1631)	3020(±1349)	4228(±2390)	3991(±514)
3078(±867)	–	4028(±1746)	–	–

number of consecutive prioritizations and convergence is reached for the highest quality levels. Independently from N_{Prio}, a high quality level is always associated to a high risk of congesting the queue. Interestingly, when N_{Prio} is very high (i.e., higher than 4), $P_{congestion}$ only depends on $clientBan_{Prio}$. This behavior is mainly caused by the third rule of the fuzzy engine (see Table 5.1), which tries to limit as much as possible a large number of consecutive prioritizations. Consequently, the segment would be considered for prioritization only when sufficient bandwidth is available in the prioritization queue. The performance of the congestion module in combination with the freeze predictor are thoroughly analyzed in Section 5.4.

5.4 Performance Evaluation Results

5.4.1 Experimental Setup

The proposed OpenFlow framework has been implemented on the Mininet Network Emulator[1]. The HTTP server, where the video content is stored, is a Jetty Server[2]. We use five different HAS videos to evaluate the proposed framework

[1] http://mininet.org
[2] http://eclipse.org/jetty

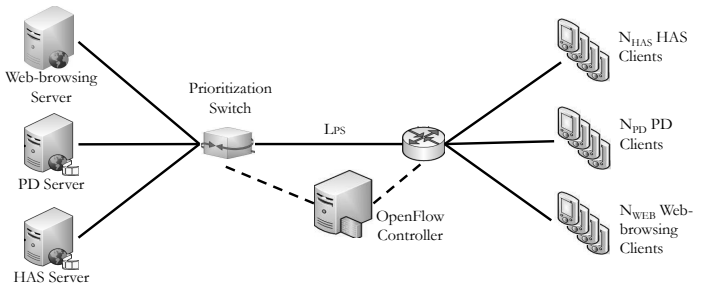


Figure 5.5: The emulated topology on Mininet is composed of several HAS, progressive download and web browsing clients, connected to the respective servers via the bottleneck link L_{PS}.

under different streaming scenarios, selected from an MPEG-DASH compliant public repository[3]. The main characteristics of the videos are summarized in Table 5.2. The videos, which belong to different genres (animation, documentary, sports and movie), are encoded in variable bitrate (VBR) and are capped to the first 10 minutes, if longer. Each video is available in a 1, 2, 4 seconds segment version, for a total of 15 different videos.

The HAS clients are implemented on top of the libdash library, the official software of the ISO/IEC MPEG-DASH standard [20]. The Libpcap library[4] is used to extract the DSCP field from the received packets and thus enable prioritization-awareness. The rate adaptation heuristic used by the HAS clients is the FINEAS algorithm[5], presented in Chapter 3. The controller is implemented using POX[6], a Python-based controller, while the OpenFlow switches are realized via Open vSwitch[7]. The prioritization switch is equipped with a best-effort and a prioritized queue. A strict-priority policy is used for the experiments, i.e., the prioritized queue can transmit at a guaranteed rate, equal to 15% of the total channel capacity.

The emulated network topology is shown in Figure 5.5, where the position of the prioritization switch is illustrated. The prioritized queue is installed on the interface towards link L_{PS}. N_{HAS} HAS clients stream the video sequence at the same time from the same HAS Server. In order to provide an extensive evaluation of the proposed framework, we emulate 10 episodes of the video trace and average the results over the 10 runs. The capacity on link L_{PS} is kept fixed during each episode while the capacity on all the other links is over-provisioned. We tested different values of the fixed capacity of link L_{PS} in order to investigate the performance of the proposed solution under different levels of network congestion.

[3] http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/
[4] http://www.tcpdump.org
[5] In this chapter, the in-network computation proposed in Chapter 3 has not been implemented.
[6] https://github.com/noxrepo/pox
[7] http://openvswitch.org

Table 5.3: Characteristics of the cross-traffic applications [21].

PD video streaming clients	**Web browsing clients**
Inter-arrival video request	*Inter-arrival page request*
Pareto distribution with mean 350 seconds and standard deviation to mean ratio of 2	Pareto distribution with mean 8.5 seconds and standard deviation to mean ratio of 1.5
Video size	*Page size*
Pareto distribution with mean 38 MBytes and standard deviation to mean ratio of 1.5	Based on web statistics [21]. Average 0.7 MBytes and standard deviation 1.5 MBytes.

Cross-traffic for HAS clients is introduced by two other types of application: N_{WEB} web browsing clients and N_{PD} PD video streaming clients. This applications mix allows to evaluate the performance of the proposed framework under realistic network conditions and to analyze the impact of prioritization both on HAS and background traffic. The implementation of the cross-traffic applications used in this chapter is obtained by following the indications presented by Akhtar et al. [21]. PD clients can stream from a single PD server, shown in Figure 5.5. Web browsing clients can download a web page from seven different web servers, which has been shown to be the average number of web hosts used to download a web page [21] (for simplicity, only one host is shown in Figure 5.5). In order to download a web page, a web browsing client opens a single TCP connection to each of the available web servers. Table 5.3 summarizes the most important characteristics of the cross-traffic applications. The inter-arrival video request time and the size of the video for PD clients are both based on a Pareto distribution, as well as the inter-arrival page request time for the web browsing clients. The web page size is based on web statistics published by Google [21], and presents an average of 0.7 MBytes and a standard deviation of 1.5 MBytes. In our experiments, the number of HAS clients, PD clients and web browsing clients is fixed to 30, 25 and 15, respectively. The percentage of video streaming traffic has been set according to the Sandvine report on Internet QoE, which shows that video streaming applications currently dominate Internet traffic [22].

In order to provide an extensive benchmark of the proposed framework, we compare our results with those obtained using three other HAS solutions. First, the FINEAS heuristic described in Chapter 3 without in-network computation, which provides the baseline for the proposed ML-based framework. Second, the Microsoft ISS Smooth Streaming (MSS) client, a popular proprietary HAS client[8]. In Chapter 4, we showed already that these two purely client-based solutions were able to achieve a low number of video freezes as well as a good video quality when compared to other heuristics. Finally, we also analyze the performance of the network-based prioritization system proposed in Chapter 4. In this case, the controller has a perfect knowledge of the status of the clients, in terms of video player buffer filling level, requested segment bitrates and video segment duration

[8]https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

Table 5.4: Characteristics of the emulated streaming scenarios.

	Video player buffer size [s]	*Segment duration [s]*	L_{PS} *[Mbps]*
Scenario 1	6	1	65
Scenario 2	10	2	60
Scenario 3	12	4	55

and can therefore fully optimize the behavior of the HAS clients. Conversely, our ML-based solution does not rely on any of this information. Comparing the performance of this solution with that of the proposed ML-based framework allows to assess the trade-off between the amount of knowledge possessed by the controller and the effectiveness of the prioritization system. The heuristic embedded in the HAS clients for both network-based solutions is the FINEAS algorithm.

5.4.2 Training the Freeze Predictor Off-line

As explained in Section 5.3.2.2, a fundamental component of the proposed prioritization framework is the freeze predictor. Based on inputs as the available bandwidth for HAS traffic, the inter-arrival time of consecutive GET requests and the requested quality, the predictor should be able to detect conditions possibly leading to a video freeze and drive the network prioritization to avoid it. The freeze predictor is based on the RUSBoost algorithm, due to its accuracy in imbalanced classification problems.

In order to train the freeze predictor, we collected a large number of HAS logs using the experimental setup reported in Figure 5.5. The network emulation setup allows us to have full control of the experiments and to collect logs about the clients' behavior after the streaming session. As an example, we can identify off-line when a client has experienced a video freeze. Consequently, we can create a training set of labeled data, which can be used to build a predictor. Each data point of the training set is associated with a segment request made by a client, and is composed of the six inputs of the predictor (see Section 5.3.2.2) and a label indicating whether the download of the requested segment resulted in a freeze or not. A machine learning algorithm can then be used to build a freeze predictor. The collected logs should be representative of the possible different video streaming scenarios, in terms of clients, network and videos configuration. For this reason, we experimented with three different streaming scenarios, reported in Table 5.4. We varied the buffer size of the clients, the segment duration of the video and the capacity of link L_{PS} (see Figure 5.5). Each scenario has been tested with all the videos reported in Table 5.2, for a total of 15 different experiments.

In total, 4500 clients' logs have been collected, resulting in almost 1.5 million individual segment requests. Only 2.5% of the segment requests are affected by a video freeze, which confirms the imbalanced nature of the analyzed classification

problem. We divided the logs into a training set, to train the predictor, and a validation set, to test its performance. Particularly, 85% of the logs collected with the *Big Buck Bunny*, *Of Forest and Man* and *Tears of Steel* videos are used to train the predictor. The remaining 15% is used as validation set. Moreover, the logs collected with the *Elephant's Dream* and *Red Bull Playstreet* videos are used as an additional validation set. This choice allows to assess to what extent the predictor can adapt to untrained videos, which is a fundamental requirement for the real deployment of the proposed approach.

We compare the performance of the RUSBoost algorithm in the freeze predictor task using four other popular classifiers: Random Forest, AdaBoost, GradientBoost and 1-Nearest Neighbourhood (1-NN) [23]. The Random Forest classifier builds an ensemble of decision trees, each contributing towards the final classification. In a random forest, N different decision trees are trained on a different sub-sample of the given dataset. Each training set is a random sample with replacement of the original dataset, and has therefore the same size as the original one. When used for classification, each tree in the random forest participates in a vote, where the class selected by the majority of the trees in the random forest is finally chosen. This process allows to increase the classification accuracy and control over-fitting, a common problem affecting standard decision trees. The AdaBoost algorithm applies the same boosting technique as the RUSBoost, without the random undersampling filtering on the training dataset. In boosting, each sample of the training set is associated with a weight. At the beginning of the training process, all the weights have the same value. At each iteration a decision tree is classified and the weights are updated. Particularly, the weights of samples that are misclassified are increased, while the weights of samples that correctly classified are decreased. This way, at each iteration, the new decision tree is forced to focus on those training samples that were misclassified by the previous decision trees. The GradientBoost classifier is a generalization of boosting methods. The number of decision trees is set to 50 for all the algorithms previously introduced. Differently from the other methods, the 1-NN classifier does not create a generalized model of the problem (e.g., the ensemble of decision trees created by the boosting algorithms). A new data point is classified with the same label as the label of the closest data point in the training set. This classifier does not require to fit a model and is typically "memory-based" (i.e., the whole training set has to be stored to classify unknown samples). Given a new point to classify, the k closest neighbors in the training set are identified, and the new point is classified using a majority vote among the k neighbors. Despite its simplicity, this method has proven to be very effective in many real classification problems. As an example, Jian et al. report how a K-NN classifier can be used to optimize the behavior of 5G networks, in terms of handover selection or energy savings [24]. The training time for the different classifiers is in the order of tens of minutes on a Dell Latitude

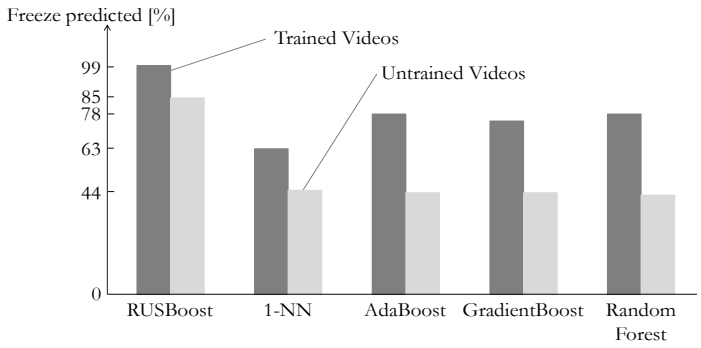


Figure 5.6: The RUSBoost algorithm is able to outperform all the other ML solutions, both for trained and untrained videos. The y-axis reports the percentage of correctly predicted freezes.

E5530 running Ubuntu 12.04 LTS 64-bit, Intel Core i5-3320M CPU @ 2.60GHz processor and 8 GB of memory. It is worth stressing that the training phase can be carried out off-line, without any real-time constraint. Only the trained model is actually used on-line in the OpenFlow controller to actively avoid the occurrence of video freezes.

The results of the ML algorithms comparison are presented in Figure 5.6. The y-axis reports the percentage of correctly predicted freezes, also called true positives. Each bar of the graph represents the percentage of true positives over the entire validation set, composed of logs from the trained videos and untrained videos. The RUSBoost classifier is able to consistently outperform the other classifiers. The freeze prediction accuracy is 99% for trained videos (*Big Buck Bunny*, *Of Forest and Man*, *Tears of Steel*) and 85% for untrained videos (*Elephant's Dream* and *Red Bull Playstreet*). Methods based on an ensemble of decision trees (AdaBoost, GradientBoost and Random Forest) are also able to reach good performance on trained videos, with almost 80% of the freezes correctly predicted. Creating an ensemble of weak classifiers helps in reducing the negative effects due to the unbalanced training set. Despite that, their performance consistently drops when exposed to untrained videos. The 1-NN classifier obtains the lowest performance[9]. In this case, new data points are classified based on the data points available in the training dataset. As the freeze prediction problem is highly unbalanced, and therefore only few examples of the positive class are present in the training dataset, this approach is prone to frequent classification errors.

Another important parameter of the freeze predictor is the percentage of non-freezes correctly predicted, i.e., the percentage of video segments not leading to a video freeze that are correctly classified as such. This metric is referred to as true

[9]Several values of k have been tested, with no significant differences in the obtained results.

Table 5.5: Summary of the off-line freeze prediction task. The percentage of correctly predicted freezes is reported, together with the corresponding percentage of predicted freeze time (between brackets).

		RUSBoost	*AdaBoost*	*Random Forest*	*1-NN*	*Gradient Boost*
Big Buck Bunny	1s	90(99)	78(85)	79(86)	64(73)	76(85)
	2s	100(100)	78(80)	78(83)	68(69)	62(68)
	4s	97(98)	49(56)	48(60)	45(70)	52(67)
Of Forest and Man	1s	100(100)	81(86)	83(87)	66(73)	79(85)
	2s	100(100)	85(86)	88(90)	68(69)	82(85)
	4s	96(99)	70(91)	60(86)	45(70)	60(86)
Tears of Steel	1s	100(100)	81(86)	83(89)	76(81)	84(87)
	2s	100(100)	73(74)	71(73)	53(50)	69(73)
	4s	98(98)	62(79)	56(79)	42(59)	51(70)
Elephant's Dream	1s	94(93)	54(62)	59(69)	51(58)	60(70)
	2s	74(69)	26(27)	21(22)	35(33)	26(29)
	4s	63(64)	17(18)	8(9)	25(24)	8(10)
Red Bull Playstreet	1s	93(81)	64(60)	70(65)	61(55)	76(69)
	2s	72(37)	32(20)	13(10)	35(18)	10(6)
	4s	92(96)	36(50)	39(59)	33(45)	37(54)

negative. As for the true positives, also the true negatives should be maximized. All the classification algorithms result in a high true negatives accuracy. The RUSBoost classifier presents an accuracy of 95%, while all the other classifiers have an accuracy higher than 98%. The 3%-4% loss of the RUSBoost classifier is largely counterbalanced by the gain in terms of true positives, which reaches 40% for untrained videos, as shown in Figure 5.6. These results therefore confirm the suitability of the RUSBoost algorithm for the freeze predictor task.

Table 5.5 reports the freeze prediction accuracy of the different classifiers, for each video. The total predicted freeze time is also reported, between brackets. This value corresponds to the total amount of freeze time associated to the correctly predicted freezes. The RUSBoost algorithm is able to outperform the other classifiers, independently of the video. As expected, higher accuracies are obtained for trained videos. The lowest performance is reached in the 2-seconds *Red Bull Playstreet* video, where only 37% of the total freeze time can be anticipated. Nevertheless, the performance of the RUSBoost algorithm is still acceptable, even for untrained videos. The AdaBoost classifier is the second best choice in terms of accuracy. As previously mentioned, the RUSBoost algorithm adds the random undersampling technique on top of the AdaBoost classifier. The 1-NN algorithm is able to achieve constant performance, both for trained and untrained videos. As it appears from Table 5.5, the freeze predictor is designed to anticipate possible video freezes, but it does not have any notion on the possible duration of these freezes. As an example, predicting 63% of the freezes in the 4-seconds *Elephant's Dream* video leads to 64% of the total freeze time predicted. This value drops to 37% in the 2-seconds *Red Bull Playstreet* video, even though the percentage of

correctly predicted freezes is 72%. In light of the above results, the RUSBoost algorithm has been chosen to implement the freeze predictor functionalities for the rest of the chapter.

5.4.3 On-line Freeze Reduction

In this section, we plan to investigate the performance of the proposed ML-based framework in terms of on-line freeze reduction. The freeze predictor trained in Section 5.4.2 is now plugged into the network controller together with the fuzzy-based congestion detection module, in order to foresee the occurrence of video freezes during an ongoing streaming session and try to actively avoid them (see Section 5.3). We compare the performance of our solution, referred to as FINEAS-ML, with that of the MSS and FINEAS clients and the network-based solution proposed in Chapter 4, called FINEAS-INF. As explained in Section 5.4.1, this solution has access to very detailed information about the video characteristics and the client's buffer filling status. The experiments have been repeated for each video and for each HAS solution, using the same streaming scenarios as in Table 5.4, for a total of 60 experiments. Each experiment has been repeated 10 times.

A high level summary of the obtained results is presented in Figure 5.7. For each experiment and for each iteration, we first compute the average number of freezes and the average freeze duration over the entire group of HAS clients. We then average these results per video. The x-axis reports the average number of video freezes obtained with the MSS, FINEAS, FINEAS-ML and FINEAS-INF solutions. The y-axis reports the average freeze duration, in seconds. As expected, the MSS and FINEAS algorithms present the worst performance, independently from the streamed video. Both algorithms are purely client-based and can fail fully optimizing the QoE of the video streaming session. Apart from the *Elephant's Dream* video (see Figure 5.7d), the FINEAS client is always able to outperform the MSS client. The number of freezes strongly depends on the video characteristics. As reported in Table 5.2, all videos have different bitrates, ranging between 254 and 4228 kbps and are encoded in VBR. In order to maintain the visual quality constant for each quality level, the bits saved during the encoding of simple scenes are reused to encode more complex scenes (e.g., with a lot of movement). Consequently, the bitrate of a specific quality level is not constant but varies depending on the video content itself. VBR encoding complicates the rate adaptation of the clients, as the segment size is not constant and it is not known in advance. As such, the videos *Of Forest and Man* and *Elephant's Dream*, which are characterized by a high variability (see Table 5.2), result in the highest amount of video freezes (Figures 5.7b and 5.7d). Particularly, the FINEAS heuristic results in the worst performance for the *Elephant's Dream* video, which presents the highest bitrate and variability for the lowest quality. The FINEAS heuristic has a

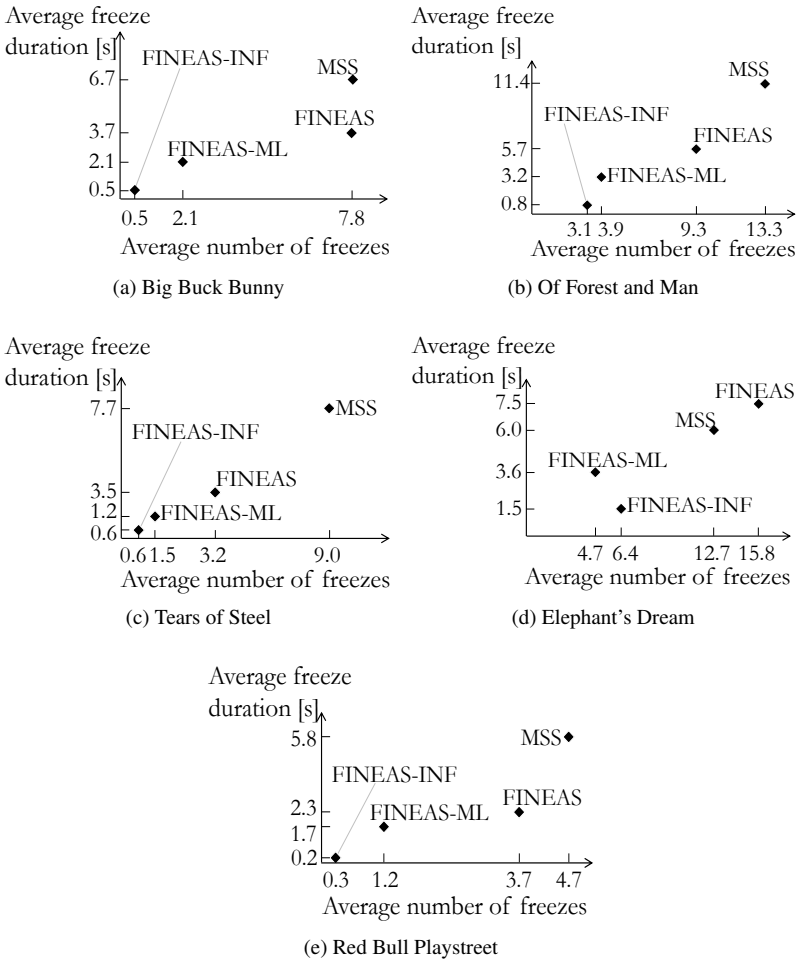


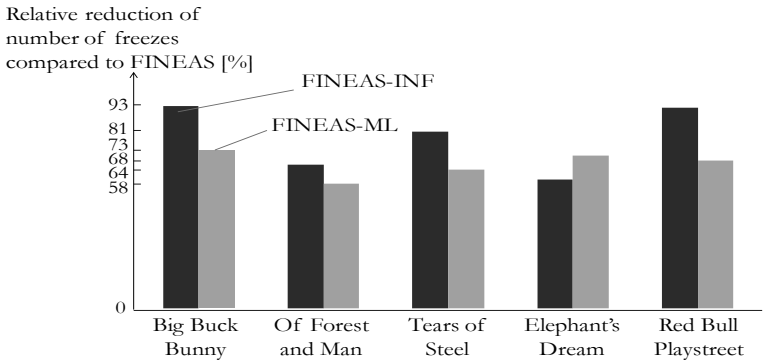
Figure 5.7: The purely client-based MSS and FINEAS heuristics present the worst performance, independently of the video. The proposed ML-based solution can consistently reduce the amount of video freezes, in all scenarios. The best performance is reached by the FINEAS-INF solution, which can use detailed information about the video and the clients' characteristics. The x-axis and y-axis report the average number of freezes and the average freeze duration, respectively.

more aggressive behavior than the MSS one and is therefore more susceptible to freezes for this specific video. Despite the high standard deviation of the bitrates of the *Tears of Steel* video (see Table 5.2), results for the FINEAS heuristic are in line with those of the *Big Buck Bunny* and *Red Bull Playstreet* videos. The nominal bitrates of the *Tears of Steel* video are the lowest or second lowest among the

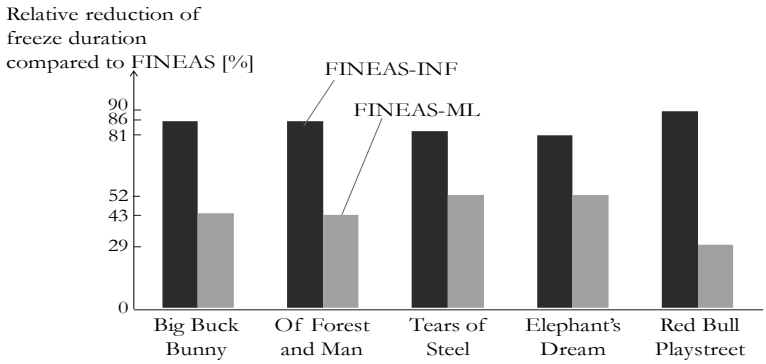
entire set, which simplifies the adaptation of the clients. The FINEAS-ML and FINEAS-INF solutions are able to outperform purely client-based solutions, both in terms of number of freezes and freeze duration. It is worth noting that the heuristic embedded in the HAS clients for both network-based solutions is the FINEAS algorithm, which represents the baseline for the performance analysis. Our proposed ML-based approach is able to consistently reduce the amount of freezes for all videos. This result also entails a strong reduction of the average freeze duration. The FINEAS-INF solution is able to reach the best performance overall and to almost completely eliminate video freezes. As previously explained, in this case the network controller has very detailed information about the characteristics of the videos (nominal bitrates and segment duration) and the status of the client's buffer and it can take the best decision in terms of segment prioritization. These results show a clear trade-off between the performance of the prioritization system and the accessibility of the inputs for the network controller. Unlike the FINEAS-INF approach, our ML-based solution does not require any special assumption on the clients' behavior, nor on the video characteristics, but it is still able to considerably reduce the amount of video freezes.

Figure 5.8 quantifies the gain brought by the proposed ML-based solution and the FINEAS-INF one, when compared to the FINEAS heuristic. Figures 5.8a and 5.8b report the relative reduction in terms of number of freezes and freeze duration, respectively, for the five analyzed videos. Despite not having access to refined information on the client's status and video characteristics, our proposed ML-based solution shows similar performance to the FINEAS-INF one, in terms of freeze reduction (Figure 5.8a). Particularly, our ML-based solution is able to reduce video freezes with about 65% compared to FINEAS, which is about 15% less than FINEAS-INF.

It is worth noting that worse performance is obtained in terms of freeze reduction compared to the results presented in Section 5.4.2. The reasons for this behavior are twofold. First, network prioritization is inherently an on-line process, while the freeze identification presented in Section 5.4.2 was performed off-line, on completed video sessions. Due to the limited resources of the prioritization queue, whose bandwidth is fixed to 15% of the total capacity of link L_{PS} (see Figure 5.5), not all the segments possibly leading to a freeze can actually be prioritized. In Section 5.3.2.3, we presented the fuzzy-based congestion detection module, whose role is to understand whether prioritizing a segment would congest the prioritization queue. When network congestion is high, many segments are identified by the freeze predictor module as in risk of a video freeze, but not all of them can be prioritized due to the decision of the congestion detection module. Moreover, as the network controller of our ML-based approach does not have any information on the video characteristics and clients' status, it results in a more conservative prioritization behavior. On average, 10% of the video is prioritized



(a) The ML-based solution is able to reduce the amount of video freezes with about 65% when compared to the FINEAS client, which is 15% less than what achieved by the FINEAS-INF solution.



(b) The proposed network-based prioritization can reduce freeze time with about 45%, even without having any information on the client's and video characteristics. The FINEAS-INF solution, which has access to this information, can reduce freeze time with about 85%.

Figure 5.8: Relative comparison between the FINEAS-INF and ML-based solutions compared to the FINEAS heuristic, in terms of number of freezes (5.8a) and freeze duration (5.8b).

in the FINEAS-ML case, while this number rises to 15% in the FINEAS-INF case. Second, bandwidth drops leading to video freezes can occur after a prioritization decision has been taken by the network controller. While the FINEAS-INF solution knows the buffer filling level of the clients and can employ a safety prioritization in this case, the same is not true for our ML-based solution, which only has limited information. This aspect also explains the better results obtained by the FINEAS-INF solution in terms of freeze time reduction, shown in Figure 5.8b. Our ML-based approach and FINEAS-INF solution can reduce freeze time with about 45% and 85% compared to FINEAS, respectively. As already reported in the

Table 5.6: Summary of the obtained results in terms of average quality (q, expressed as an integer between 0 and the highest quality level, which varies between 5 or 6 depending on the video), number of freezes (NF) and freeze time (FT, in seconds). Differences between cells associated with the same uppercase letter are not statistically significant (t-test, $p \leq 0.05$).

		FINEAS(a)			*FINEAS-ML(b)*			*FINEAS-INF(c)*			*MSS(d)*		
		q	NF	FT[s]	q	NF	FT[s]	q	NF	FT[s]	q	NF	FT[s]
Big Buck Bunny	1s	2.98	17.4	7.33	2.95	4.31	3.69^{d}	2.97	0.59	0.13	3.1	9.54	3.02^{b}
	2s	2.84	4.83^{d}	2.52^{d}	2.68	0.68	0.91	3.02	0.02	0.01	2.79	6.02^{a}	3.42^{a}
	4s	2.18	1.16^{b}	1.26^{b}	2.08	1.31^{a}	1.71^{a}	2.05	0.84	1.41	2.14	7.93	13.7
Of Forest and Man	1s	2.72	16.93^{d}	7.43^{d}	2.47	6.40	3.69	2.67	8.62	0.97	2.70	15.80^{a}	6.49^{a}
	2s	2.63	8.32^{d}	4.66^{d}	2.36	2.54	2.29	2.24	0.04	0.01	2.60	7.18^{a}	5.18^{a}
	4s	2.06	2.77^{b}	4.99^{b}	1.85	2.88^{a}	3.76^{a}	1.84	0.74	1.39	1.94	16.83	22.47
Tears of Steel	1s	2.90	5.04	2.80^{d}	2.78	1.62	1.31	2.74	0.96	0.22	2.64	8.96	3.25^{a}
	2s	2.67	2.76	1.63	2.61	0.29	0.39	2.69	0.03	0.01	2.60	5.50	4.77
	4s	2.03	2.66	5.21	1.96	1.81	2.94	1.90	0.90	1.56	2.01	8.78	19.11
Elephant's Dream	1s	2.07	27.60^{d}	10.39^{d}	1.94	6.69	3.92	2.04	16.91	3.31	2.05	24.69^{a}	8.52^{a}
	2s	1.89	9.51	4.92	1.79	3.25	2.83^{d}	1.88	0.68	0.24	1.68	5.73	3.32^{b}
	4s	1.59	10.27^{d}	7.24^{d}	1.51	4.03	4.11	1.52	1.65	1.05	1.69	7.81^{a}	6.22^{a}
Red Bull Playstreet	1s	2.27	8.80	3.45	2.07	1.58	1.53^{d}	2.15	0.31	0.05	2.16	4.57	1.35^{b}
	2s	2.18	0.76^{b}	0.83^{b}	2.05	1.05^{a}	$2.08^{a,d}$	2.03	0.08	0.01	1.85	2.43	1.85^{b}
	4s	1.64	1.69	2.75	1.52	0.91	1.38	1.53	0.40	0.59	1.68	7.07	14.36

previous section, the freeze predictor based on the RUSBoost algorithm is designed to reduce the amount of video freezes. This condition however does not always result in a correspondent freeze time reduction.

Table 5.6 reports the complete results, for each video and for each different version in terms of segment duration. We report the average requested quality, number of freezes and freeze duration. We also perform an analysis to assess

whether the differences between the different heuristics are statistical significant or not (t-test, $p \leq 0.05$). Each heuristic is assigned a letter, from *a* for FINEAS to *d* for MSS. Results associated with one of these letters are not statistically different from the results obtained by the heuristic associated with the correspondent letter. As an example, if a result of the FINEAS heuristic is associated with the letter *d*, then it is not statistically different from the result obtained by the MSS client, and viceversa. Overall, the results of the FINEAS-INF and FINEAS-ML approaches show strong statistical significance when compared to the FINEAS and MSS heuristics. The FINEAS-INF solution is able to almost completely eliminate video freezes, in all video streaming scenarios. Only in the 1-second segment version of the *Of Forest and Man* and *Elephant's Dream* videos, this approach shows sub-optimal performance. In the FINEAS-INF approach, one of the inputs of the controller are the nominal bitrates of the streamed video. As reported in Table 5.2 though, these two videos present a high bitrate variability. Moreover, encoding the video with 1-second segments results in a slightly higher nominal bitrate (about 5%) compared to the 2-seconds version, due to the cost of more frequent I-frames at the beginning of the segments. These disturbances complicate the task of the controller.

Thanks to the RUSBoost and fuzzy algorithms, our ML-based approach is able to reduce video freezes when compared to the baseline FINEAS heuristic, without needing any explicit information on the video or client's configuration. Our approach presents worse performance compared to the FINEAS heuristic only in two cases (4-seconds *Big Buck Bunny* and 2-seconds *Red Bull Playstreet*), but these differences are not statistically significant. Similar conclusions can also be drawn for the MSS heuristic. These results again confirm the suitability of the proposed approach for the on-line freeze reduction task. Another important aspect to analyze is the requested video quality. In order to maximize users' QoE, a heuristic should be able to not only avoid freezes, but also to request the highest possible video quality. As reported in Table 5.6, only minor differences can be noted among the different HAS solutions. The average quality slightly decreases (with about 5%) in the FINEAS-INF and ML-based cases when compared to the FINEAS heuristic. This result is mainly due to the prioritization mode, where a client is forced, by design, to request the lowest quality level in case the most recently downloaded segment has been prioritized. It is also worth mentioning that the quality tends to decrease when longer segments are used. In this situation, the clients have less fine-grained decision points to adjust the quality to the available bandwidth, consequently resulting in a more conservative behavior. This condition is especially true when the maximum buffer size of the clients is limited to a few seconds in order to reduce the end-to-end delay in live video streaming scenarios, as in our experiments.

Another important aspect to consider is the distribution of video freezes among the different clients. Ideally, the freeze reduction reported in the previous para-

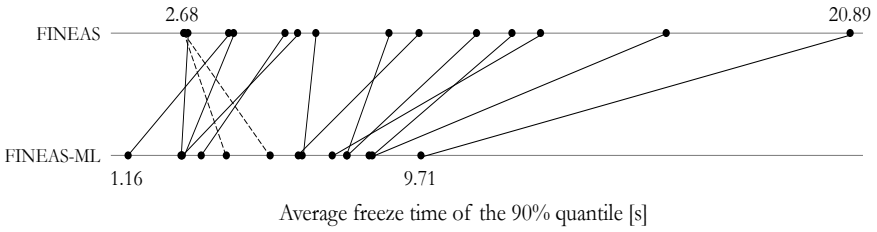


Figure 5.9: Our ML prioritization framework has a consistent impact on the freeze distribution, by reducing freeze time for all the clients. The figure reports the average freeze time of the 90% quantiles, for all the possible network and video configurations.

graphs should be equally shared among the different clients, in order to provide fairness to the entire system. To show this behavior, we computed the 90% quantile of the average client freeze time, for all the different network and video configurations as reported in Table 5.6. The 90% quantile expresses the maximum freeze time experienced by 90% of the clients. By using the proposed ML prioritization framework, we should be able to reduce not only the average freeze time (as shown in Figure 5.8) but also its 90% quantile, which would entail that all the clients benefit from the prioritization system. The results of this analysis are reported in Figure 5.9, where the 90% quantile of all the 15 different streaming configurations are reported, for both FINEAS and FINEAS-ML. Using our prioritization framework results in a consistent reduction of the 90% quantile of the average freeze time, when compared to the purely client-based FINEAS heuristic. As explained before, this reduction entails that all the HAS clients benefit from prioritization. In only two configurations the 90% quantiles increase, which correspond to the streaming scenarios where the 4-seconds *Big Buck Bunny* and 2-seconds *Red Bull Playstreet* videos are used (see dotted lines in Figure 5.9). As reported in Table 5.6, in these two cases the average freeze time is higher when prioritization is used (even though this difference is not statistically significant).

5.4.4 Heterogeneous Clients

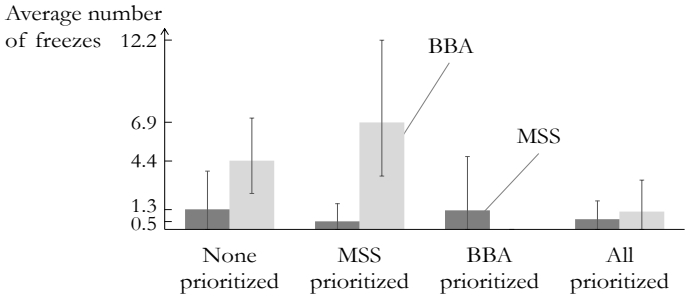
In this section, performance results are shown of the prioritization framework when different rate adaptation heuristics are used instead of the FINEAS algorithm. As reported in Section 5.4.2, the freeze predictor based on the RUSBoost algorithm has been trained using logs generated by FINEAS clients only. Despite that, the prioritization system should provide good performance even when different clients are equipped with different heuristics, a common situation in real deployments. The same topology as in Figure 5.5 is used for the experiments, with capacity on link L_{PS} fixed to 65 Mbps. The clients stream the 1-second version of the *Big Buck Bunny* video.

Table 5.7: Summary of the results when the BBA algorithm is used as adaptation heuristic, in terms of average quality (q, expressed as an integer between 0 and 6), number of freezes (NF) and freeze time (FT, in seconds). The 10% and 90% quantiles are reported between brackets.

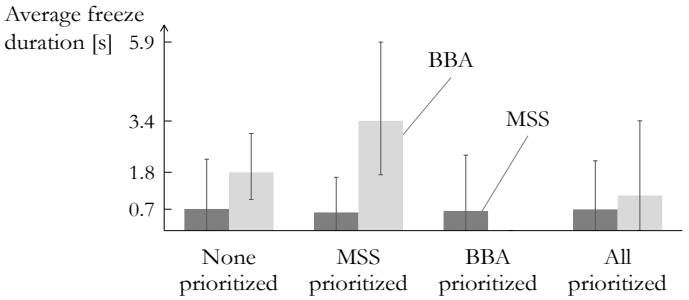
	q	NF	FT[s]
BBA	3.2(2.4-3.9)	4.6(1.5-9.0)	1.9(0.5-3.8)
BBA-ML	2.8(2.4-3.2)	2.0(0.3-3.6)	1.8(0.1-3.4)

In a first set of experiments, all HAS clients use the BBA algorithm by Huang et al. [1]. The BBA client is an example of a purely buffer-based algorithm, where the next quality is selected based on the buffer level only. Particularly, Huang et al. showed on a real field test that the BBA client is able to outperform the standard Netflix player. This choice allows to assess the performance of the prioritization system when the clients are equipped with a completely different heuristic compared to the one used for training. The main results are reported in Table 5.7, for a buffer size fixed to 20 seconds. The proposed ML-based framework can consistently reduce the number of freezes by 57%. It is worth stressing that this result is achieved even though the freeze predictor module has been trained with a different adaptation heuristic. Even though freezes can be limited, the average freeze time is similar for both the BBA and BBA-ML solutions. The prioritization system is effective in avoiding short freezes, while long freezes are more difficult to predict because of the different behavior of the purely buffer-based BBA heuristic compared to the FINEAS heuristic used for training. As expected, the video quality is slightly reduced when prioritization is enforced, since the clients are forced to request the lower quality in case of prioritization.

In a second set of experiments, we investigate a heterogeneous scenario, where 50% of the clients are equipped with the BBA heuristic and 50% with the MSS one. We analyze the performance of the system both when all the clients are prioritized and when only one group can benefit from prioritization. The same experimental setup as for the previous experiment has been used. The correspondent results are shown in Figure 5.10, in terms of average number of freezes and freeze duration. For each emulated iteration, we also computed the 10% and 90% quantiles, which quantify the maximum number of freezes and freeze time experienced by 10% and 90% of the HAS clients, respectively. Each point of the graphs is therefore associated with the average 10% and 90% quantiles over the 10 iterations. When none of the clients is prioritized, the BBA heuristic exhibits worse performance than the MSS one, both in terms of number of freezes ($\times$3.4) and freeze duration ($\times$2.6). This behavior is due to the purely buffer-based nature of the BBA client, which does not take into account the available bandwidth and is therefore more susceptible to freezes unless a very large buffer is used. When only the MSS group is prioritized, freezes can be reduced by 62% (see Figure 5.10a) for MSS clients,



(a) Average number of freezes.



(b) Average freeze duration.

Figure 5.10: Even in a heterogeneous scenario, our system can always reduce the number of freezes for the prioritized clients. When only one group is prioritized, the influence on the non-prioritized group depends on the underlying heuristic.

while freeze duration is similar to the non-prioritized case (see Figure 5.10b). Interestingly, prioritizing the MSS group has a negative impact on the performance of the BBA group, as both the number of freezes and freeze time increase when compared to the non-prioritized case. When prioritization is enforced by the controller, the bandwidth available for the clients in the best-effort queue can drop. As BBA clients do not consider this metric in their adaptation, they are affected by more video freezes. When only the BBA group is prioritized, freezes can be completely eliminated, for all BBA clients. This result does not negatively affect the performance of the MSS clients, which consider both the buffer level and the available bandwidth in their quality adaptation, and can therefore react quickly to changing bandwidth conditions. When all the clients can benefit from prioritization, freezes are reduced for both groups. Even though the ML-based prioritization system is trained in a scenario where all the clients are equipped with the FINEAS algorithm, the learned client model can be effectively re-used to optimize the behavior of other adaptation heuristics, even in heterogeneous scenarios.

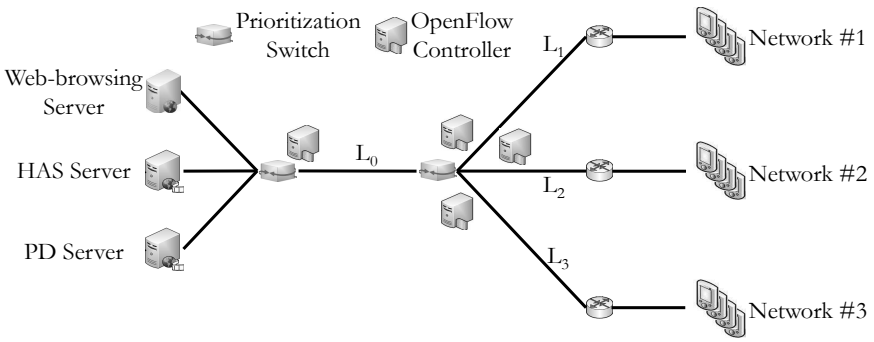
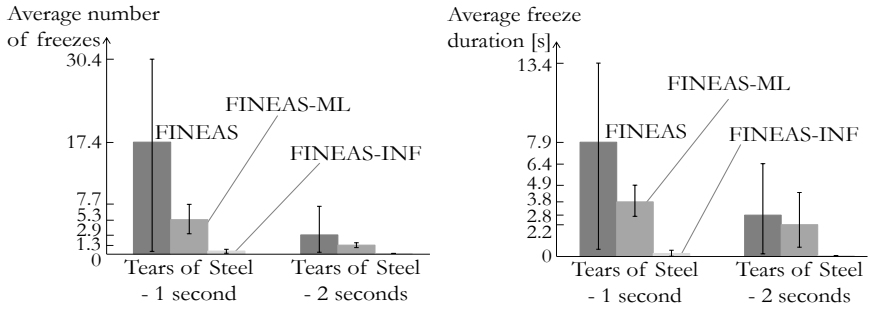


Figure 5.11: Three networks compose the topology emulated on Mininet. Links L_0, L_1, L_2 and L_3 are the bottlenecks and are equipped with a prioritization queue. Each link is associated with an independent OpenFlow controller.

5.4.5 Multiple Bottlenecks Network Topology

So far, we have investigated a single bottleneck scenario, where congestion can occur on one link only (i.e., link L_{PS} in Figure 5.5). In real networks however, bottlenecks can arise on different links simultaneously and at different network levels. The goal of this section is therefore to investigate the performance of the proposed ML-based solution in a multi-bottleneck network scenario. In this scenario, each possible bottleneck should be associated with an independent network controller, equipped with the same set of algorithms as described in Section 5.3.2. It is important to stress that no communication should be envisioned among the independent controllers, as this approach would require extra-signaling inside the network. Despite that, the controllers should still be able to achieve a global coordinated behavior in order to provide end-to-end prioritization. This distributed approach presents two advantages with respect to a centralized one, where one single controller is responsible for all the network bottlenecks. First, it is inherently more scalable. In Chapter 4, we showed that the proposed prioritization system can control up to 5000 HAS clients simultaneously. The controller could therefore be overloaded when a centralized solution is used. Second, a distributed approach is easier to deploy, as no modifications to the design presented in Section 5.3.2 are required. Even in a multi-bottleneck scenario, prioritization-awareness using the DSCP field can still be guaranteed at the client. Indeed, the switches can only set the DSCP bit to 1 when the segment is prioritized, while it is left unaltered in case of a best-effort delivery. It is worth noting that the controllers associated to the switches are the only responsible to decide about prioritization: the DSCP field is only used to communicate a prioritization event to the client and does not automatically trigger prioritization in the switches.

In light of the above, the multi-bottleneck topology shown in Figure 5.11 has



(a) Average number of freezes

(b) Average freeze duration

Figure 5.12: The proposed ML-based approach can reduce the amount of freezes and the freeze time with respect to the FINEAS heuristic, even in a multi-bottleneck scenario with independent controllers. The FINEAS-INF solution achieves the best results overall.

been emulated on Mininet. The HAS, PD and web browsing clients are divided into three networks. Each network is composed of 10 HAS clients equipped with the FINEAS heuristic, 8 PD clients and 5 web-browsing clients, for a total of 69 clients. Links L_0, L_1, L_2 and L_3 represent the possible bottlenecks for the HAS clients. Each link is equipped with a prioritization queue, with bandwidth equal to 15% of the channel capacity, and is controlled by an independent network controller. Even though the controllers for links L_1, L_2 and L_3 can be located together, we decided to logically split them in order to show the performance of the system in a fully distributed scenario. Two different video streaming scenarios have been evaluated. In the first one, the HAS clients are equipped with a 6 seconds buffer and stream the 1-second segment version of the *Tears of Steel* video. Capacity on links L_0 and $L_{1\text{-}3}$ is fixed to 75 Mbps and 30 Mbps, respectively. In a second scenario, the 2-seconds segment *Tears of Steel* video is streamed, and the clients use a 10 seconds buffer. L_0 has a capacity of 70 Mbps, while links $L_{1\text{-}3}$ have a capacity of 28 Mbps. Each experiment has been repeated 10 times. Depending on the cross-traffic on links L_1, L_2 and L_3, the actual bottlenecks for the three networks dynamically change. This way, we can explore a wide range of network configurations, where the three networks could possibly influence each other.

Figure 5.12 reports the obtained results for the FINEAS, FINEAS-ML and FINEAS-INF approaches, in terms of average number of freezes and average freeze duration. Each point of the graphs is also associated with the average 10% and 90% quantiles over the 10 iterations. The client-based FINEAS heuristic results in the highest amount of freezes and freeze time. The presence of congestion at multiple network levels makes it harder for the client to avoid freezes, which are up to 3 times higher than in the single bottleneck case. Moreover, some of the clients experience very high freeze time (up to 13.4 seconds), as the quantiles

indicate. Network-based prioritization can consistently improve the performance of the system, both for the proposed FINEAS-ML approach and the FINEAS-INF one. Even though independent, the controllers are able to take coordinated actions on which client to prioritize. This behavior is due to the type of inputs used by the controllers to decide on prioritization. Particularly, each controller can obtain *local* information about the status of the controlled link and *global* information about the HAS clients. Information about the clients is composed of the quality level of the requested segment and the time between consecutive HTTP GET requests, as described in Section 5.3.2.2. The status of the controlled link is a local information that is only available to the specific controller. Conversely, the requested quality and the GET inter-arrival time are measurements that can be obtained by all controllers, independently of their position. Consequently, the controllers are fed with some inputs that are global and shared with all the other controllers. This specific condition facilitates coordination among the otherwise independent controllers. As expected, the FINEAS-INF solution is able to reach the best performance, both in terms of video freezes and freeze duration. Our ML-based solution can reduce video freeze with 70% in the 1-second segment *Tears of Steel* video and 55% in the 2-seconds segment version (Figure 5.12a), when compared to FINEAS. These results correspond to a 50% and 13% freeze time reduction (Figure 5.12b). The low dispersion indicated by the 90% quantiles also shows that all the clients obtain similar performance and confirms the effectiveness of the proposed prioritization system. It is worth stressing that these results are obtained in a completely distributed scenario, where the controllers do not communicate among each other, and without using any information on the streamed video or on the clients' status. Despite that, the proposed ML-based approach can still provide good performance, even in a multi-bottleneck network scenario.

5.5 Conclusions

We presented in this chapter a novel network-based framework to prevent the occurrence of video freezes for HAS clients. The main element of this framework, implemented using the OpenFlow protocol, is a network controller. This controller can prioritize the delivery of video segments likely leading to a freeze using a dedicate queue. Prioritization is driven by a machine learning engine, based on the RUSBoost algorithm and fuzzy logic. The RUSBoost algorithm is used to detect whether a client is close to a freeze, while fuzzy logic allows to understand whether the conditions of the prioritization queue are good enough to successfully prioritize the segment. Compared to the prioritization logic presented in Chapter 4, no knowledge on the video, in terms of bitrates and segment duration, is required in this case, nor on the client's configuration, in terms of initial buffering time. This aspect simplifies the practical applicability of the proposed framework in a real

deployment. Results obtained through emulation showed that our ML-based approach can consistently reduce video freezes with about 65% and freeze time with 45%, when compared to the benchmarking heuristics FINEAS and MSS. Moreover, the proposed approach has also been evaluated in a multi-bottleneck network scenario, where we showed that a system of distributed independent controllers is still able to reduce the amount of video freezes with about 60%.

The prioritization algorithms designed in Chapter 4 and 5 present an alternative solution to the same problem. Both solutions have advantages and disadvantages. The approach described in Chapter 4 is able to reach very good performance, but this requires to obtain specific information on the streamed video and the client's configuration, which might be difficult to collect in practice. Moreover, the algorithm requires a certain degree of fine tuning to be able to work properly. The approach described in this chapter relies instead on less information and requires less tuning. Nevertheless, this results in a smaller gain compared to the approach in Chapter 4. When flexibility is the most stringent requirement for the deployment of the prioritization framework, then the approach presented in Chapter 5 is clearly the best solution: it requires less human tuning and it can improve over time, as it involves a learning process. Moreover, the ML-based approach can be extended to work also in case of encryption. In this case, the quality level input might not be available for the freeze predictor. Two possible countermeasures can be adopted in this case. First, the freeze predictor can be re-trained without considering the quality level as an input; while this would reduce the performance of the system, it would also guarantee that the resulting predictor would work with encrypted traffic. Second, the quality level itself can be estimated online, for example using an additional ML algorithm, as shown by Dimopoulos et al. [25]. Even though more complex, this approach would allow to keep the freeze predictor as is. The algorithm presented in Chapter 4 is instead more difficult to extend to cope with encryption, as it requires several information about the video and the HAS clients. On the other hand, if the video delivery is fully managed, which entails that the video traffic is not encrypted and it is possible to collect the extra information required for the algorithm in Chapter 4, then this solution represents the best approach, as it allows to reach the best performance.

Acknowledgment

The research was performed partially within the strategic research project SMILE-IT funded by VLAIO and the iMinds PRO-FLOW project (under VLAIO grant agreement no. 150223). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

References

[1] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. *A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service*. In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 187–198, New York, NY, USA, 2014. ACM.

[2] CONVIVA. *2015 Viewer experience report*. http://www.conviva.com/conviva-viewer-experience-report/vxr-2015/.

[3] R. Masoudi and A. Ghaffari. *Software defined networks: A survey*. Journal of Network and Computer Applications, 67:1 – 25, 2016.

[4] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. *RUSBoost: A Hybrid Approach to Alleviating Class Imbalance*. Trans. Sys. Man Cyber. Part A, 40(1):185–197, January 2010.

[5] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials 99, 2014, 2014.

[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. *A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP*. SIGCOMM Comput. Commun. Rev., 45(4):325–338, August 2015.

[7] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. *Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale*. IEEE Journal on Selected Areas in Communications, 32(4):719–733, April 2014. doi:10.1109/JSAC.2014.140405.

[8] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive*. Networking, IEEE/ACM Transactions on, 22(1):326–340, Feb 2014.

[9] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. *CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction*. In Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, pages 272–285, New York, NY, USA, 2016. ACM.

[10] K. Ivesic, L. Skorin-Kapov, and M. Matijasevic. *Cross-layer QoE-driven admission control and resource allocation for adaptive multimedia services in LTE*. Journal of Network and Computer Applications, 46:336 – 351, 2014.

[11] A. Ganjam, J. Jiang, X. Liu, V. Sekar, F. Siddiqi, I. Stoica, J. Zhan, and H. Zhang. *C3: Internet-scale Control Plane for Video Quality Optimization*. In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, pages 131–144, Berkeley, CA, USA, 2015. USENIX Association.

[12] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. *Practical, Real-time Centralized Control for CDN-based Live Video Delivery*. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, pages 311–324, New York, NY, USA, 2015. ACM.

[13] H. Egilmez, S. Civanlar, and A. Tekalp. *An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks*. IEEE Transactions on Multimedia 15, 2013, April 2013.

[14] T. Uzakgider, C. Cetinkaya, and M. Sayit. *Learning-based approach for layered adaptive video streaming over SDN*. Computer Networks, 92, Part 2:357 – 368, 2015.

[15] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. *Measuring Home Broadband Performance*. Communications of the ACM, 55(11):100–109, November 2012.

[16] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming*. In Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[17] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. *Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 3:1–3:12, New York, NY, USA, 2016. ACM.

[18] T. Wu, R. Huysegems, and T. Bostoen. *Scalable network-based video-freeze detection for HTTP adaptive streaming*. In 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), pages 95–104, June 2015.

[19] H. R. Berenji. *A Reinforcement Learning-Based Architecture for Fuzzy Logic Control*. International Journal of Approximate Reasoning, 6(2):267 – 292, 1992.

[20] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer. *Libdash - An open source software library for the MPEG-DASH standard*. In Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on, pages 1–2, July 2013.

[21] S. Akhtar, A. Francini, D. Robinson, and R. Sharpe. *Interaction of AQM schemes and adaptive streaming with Internet traffic on access networks*. In Global Communications Conference (GLOBECOM), 2014 IEEE, pages 1145–1151, Dec 2014.

[22] Sandvine. *Exposing the Technical and Commercial Factors Underlying Internet Quality of Experience*. https://www.sandvine.com/trends/global-internet-phenomena/.

[23] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, New York, 2009.

[24] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, and L. Hanzo. *Machine Learning Paradigms for Next-Generation Wireless Networks*. IEEE Wireless Communications, PP(99):2–9, December 2016.

[25] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki. *Measuring Video QoE from Encrypted Traffic*. In Proceedings of the 2016 Internet Measurement Conference, IMC '16, pages 513–526, New York, NY, USA, 2016. ACM.

6

A Scalable WebRTC Framework for Remote Video Collaboration Applications

S. Petrangeli, D. Pauwels, J. van der Hooft, M. Žiak, J. Slowack, T. Wauters and F. De Turck.

Submitted to Multimedia Tools and Applications, December 2017.

⋆ ⋆ ⋆

The previous chapters have focused on the optimization of HTTP Adaptive Streaming (HAS) solutions, which are the dominant technology in live streaming and Video-On-Demand (VoD) scenarios. The goal of this chapter is to provide a different perspective, by considering those streaming cases that require low-latency and interactivity, as remote conferencing, telehealth and remote teaching applications. In this case, real-time communication solutions based on RTP are generally used instead of HTTP-based solutions. Particularly, the Web Real-Time Communication (WebRTC) protocol is of extreme interest in this case, as it allows any two or more remote peers equipped with a browser to communicate among each other, without the need of external plugins. WebRTC is by design peer-to-peer, which means that each sending peer needs to have a dedicated encoder for

each receiving peer. To increase the cost-efficiency and scalability of this architecture, this chapter proposes a framework where only a limited number of encoders are used by the sending peers. A centralized node, called conference controller, receives the streams encoded by the sender and forwards them to the receivers, based on their bandwidth conditions. As in this case multiple receiving peers are associated with the same encoder at sender-side, the conference controller periodically recomputes the encoding bitrates to follow the long-term network conditions of the receivers. The gains brought by the proposed framework have been confirmed in both simulation and emulation, through a testbed implemented using state-of-the-art WebRTC software. In an emulated scenario where a single sending peer equipped with three encoders transmits to 28 receivers, the proposed framework improves the average received video bitrate up to 15%, compared to a static solution where the encoding bitrates do not change over time.

6.1 Introduction

Remote video collaboration is widely used in a variety of applications nowadays [1–3]. From videoconferencing, to telehealth and remote teaching, it allows to remotely perform tasks that would otherwise require a physical meeting, and it is therefore an important enabler for fast and efficient exchange of information. Remote collaboration communication can be roughly divided in two categories, depending on the type of interaction established among the remote participants. In many-to-many communication, all the remote participants have the same importance and frequently interact among each other. In one-to-many communication instead, the interaction among the participants is usually dominated by a single entity. A classical example of such one-to-many communication is represented by remote teaching applications. In this *virtual classroom*, the students remotely attend a live lecture given by the lecturer. Interactivity is required in this case, as the students can ask questions and actively participate to the discussion. Nevertheless, most of the communication occurs from the lecturer to the students. In both many-to-many and one-to-many scenarios, the remote peers are usually geographically distributed and can experience different bandwidth and network conditions.

Classical streaming techniques as HTTP adaptive streaming are characterized by high latencies, in the order of seconds in the best case [4], and can therefore not guarantee the required degree of interactivity of remote video collaboration applications. On the other hand, remote conferencing solutions can be used. Particularly, the Web Real-Time Communication (WebRTC) framework is an open-source project started by Google in 2011 that provides plugin-free real-time communication capabilities to browser-based applications [5]. Even though this technology guarantees the low-latency and interactivity degree required in remote collaboration, it is affected by another drawback. The WebRTC framework has been devel-

oped with a peer-to-peer architecture in mind, where a small group of clients can directly communicate with each other. This approach can suffer from scalability issues when many participants are present at the same time. In standard WebRTC indeed, the peers in communication, or *senders*, would need to encode a separate stream for each receiving peer, the *receivers*. This aspect entails that each receiver is associated with an independent and dedicated encoder at sender-side. This architecture is particularly inefficient in a one-to-many scenarios, as in this case, a single peer is usually responsible for the largest part of the communication.

In order to reduce the scalability issue of such a peer-to-peer architecture, we propose a WebRTC-compliant framework to support the delivery of real-time communication streams in a one-to-many remote video collaboration scenario, as the virtual classroom. In this framework, the WebRTC sender only needs to encode a limited number of streams, much smaller than the number of receivers, at different bitrates. This approach allows to overcome the aforementioned limitation, where each receiver would need to be associated to an independent, dedicated encoder. In our framework, instead, multiple receivers are assigned to the same encoder at sender-side. A centralized node, called the *conference controller*, is aware of the bandwidth conditions of the WebRTC receivers and dynamically forwards the stream at the best bitrate in order to follow the bandwidth variations of the receivers. Besides this dynamic stream forwarding, the conference controller has another fundamental task. Instead of keeping the encoding bitrates of the sender fixed to predefined static values, the conference controller can periodically recompute them based on the changing bandwidth conditions of the receivers. This approach allows to better follow the bandwidth conditions of the receivers, even though only a limited number of encoders is actually used. It is worth noting that the goal of the proposed framework is not to optimize the WebRTC protocol itself, which already guarantees the required interactivity of remote collaboration applications, but rather relieving its scalability problems using the conference controller. The controller mainly optimizes the delivery of the real-time streams from sender to receivers, as in a one-to-many scenario most of the communication follows this path. In the WebRTC domain, the conference controller functionalities can be carried out by a Selective Forwarding Unit (SFU), whose task is to receive all the streams and decide which stream should be sent to which participant [6]. Particularly, the Jitsi-Videobridge software is used as WebRTC SFU[1].

The contributions of this chapter are threefold. First, we present in detail the proposed framework and the conference controller functionalities. Particularly, we model the bitrate recomputation problem as an Integer Linear Programming (ILP) formulation, which is periodically solved by the centralized node. We also propose a fast and scalable algorithm, using the K-means clustering algorithm, to solve the aforementioned problem in an approximate way when the number of

[1] https://github.com/jitsi/jitsi-videobridge

receivers is too large. Second, we present an emulation testbed, implemented using the WebRTC protocol suite and Jitsi-Videobridge, to evaluate the performance of the proposed framework in a realistic environment. Third, detailed results are presented to quantify the gains brought by the proposed approach. Particularly, simulation results are presented to theoretically evaluate the performance of the WebRTC framework in a large number of configurations. The emulation testbed is then used to confirm these results in a realistic setting.

The remainder of this chapter is structured as follows. Section 6.2 presents related work on remote conferencing solutions for WebRTC. Section 6.3 introduces the general architecture of a WebRTC-based communication system, with a particular focus on the bandwidth estimation and congestion control performed by WebRTC endpoints. The architecture of the proposed framework is described in Section 6.4. The functionalities of the conference controller and the ILP formulation for the dynamic encoding bitrate recomputation are presented in detail in Section 6.5, while Section 6.6 details the implementation of the emulation testbed. An in-depth analysis of the proposed framework, by means of obtained simulation and emulation results, is presented in Section 6.7. Finally, Section 6.8 concludes the chapter.

6.2 Related Work

Xu et al. perform a measurement study on real-world conferencing systems [7]. The authors report that a purely peer-to-peer architecture is not popular among these systems, as it does not scale to a large number of users. To improve the scalability of this architecture, an intermediate media server can be used. In WebRTC, this can be achieved using two different components: a Multipoint Conferencing Unit (MCU) or a selective forwarding unit.

An MCU receives all the streams from the participants, decodes and composes them in a single common stream that is sent back to the peers. This way, each peer only needs to send and receive a single stream. Two popular MCU implementations are already available, the Janus gateway and Kurento [8, 9]. Janus is conceived as a general purpose gateway that only allows, in its core functionality, to setup a WebRTC communication among the peers [8]. Higher level functionalities are implemented as Janus plugins, as the video MCU plugin, which implements an MCU and supports both many-to-many and one-to-many communication scenarios. Kurento is another example of WebRTC MCU, which also provides advanced functionalities as computer vision and augmented reality [9]. A set of engineered and coherent APIs are also available for the control of the media server, to allow developers to quickly deploy new functionalities [10]. Ma et al. investigate how to improve the encoding/decoding process of an MCU to save bandwidth [11]. The MCU transcodes the sender stream and adjusts it to the viewing conditions of the

receiver. The authors consider the viewing distance and the pixel density of the receiver's screen to transcode the stream to an optimal bitrate, in order to save bandwidth. A network-wide system of MCUs is investigated by Grand et al. [12]. The participants are divided into regional clusters, each associated to an MCU. Peers located in different clusters can communicate via the system of MCUs, connected among each other using a peer-to-peer network. This hybrid architecture allows to support a large number of users. Nevertheless, MCU operations are extremely computationally intensive, due to the decoding-mixing-encoding processes that have to be carried out. To reduce this issue, MCU functionalities can be dynamically migrated among conference participants to meet certain bandwidth, latency and CPU constraints [13]. Alternatively, MCU low-level functionalities can be virtualized and deployed on-the-fly [14, 15]. As an example, Rodriguez et al. divide the low-level functionalities of an MCU into independent broadcasters, which can run in distributed environments [16].

Unlike an MCU, an SFU does not require decoding/encoding operations and it is therefore more lightweight. Its main task is to receive all the streams from the participants and selectively forward one or more streams to the peers [17]. When the number of participants is large, the amount of forwarded streams should be limited to avoid wasting bandwidth. For this reason, Grozev et al. develop a speaker identification algorithm to be deployed on an SFU, to identify the last N dominant speakers of the conference [6]. To save bandwidth, only these N streams are forwarded to the conference participants. In a measurement study, Xhagjika et al. find that the load pattern on an operational system of SFUs is periodic and can be easily predicted [18]. The prediction can be used to allocate the streams to the right SFU and avoid overloading the system. The software-defined networking principle can be used to optimize a system of distributed SFUs, by dynamically creating a multicast tree for the optimal delivery of the streams [19]. The functionalities of an SFU can be matched with the concept of simulcast in WebRTC [20]. In simulcast, each sending peer encodes the stream at different bitrates, which are then forwarded to the receivers by the SFU. In the work by Grozev et al. [20], each participant can send up to three streams. The SFU forwards the highest quality to participants involved in the conversation, and the lowest quality to the remaining ones. The authors also point out that simulcast is one of the less mature parts of the WebRTC standard, and that it still needs further development and optimizations.

In this chapter, the conference controller is implemented using the SFU functionalities. Unlike previous works though, the controller not only forwards the streams to the remote peers, but also periodically recomputes the set of encoding bitrates of the sending peer. This bitrate recomputation is modeled as an ILP formulation, which can be optimally solved when the number of receivers is small. Otherwise, an approximate solution is obtained using the K-means clustering algo-

rithm. A similar formulation of the encoding bitrate recomputation has also been presented by De Praeter et al., for an HTTP adaptive streaming scenario [21]. The work presented in this chapter goes several steps further by designing a complete WebRTC-compliant framework to actually evaluate the performance of the system, and a simulator, which takes into account the characteristics of WebRTC to test the performance of the proposed recomputation algorithms.

A preliminary evaluation of the proposed framework has already been presented in previous work [22]. This chapter provides a more detailed explanation of the conference controller functionalities, supported by extensive simulation and emulation results to prove its effectiveness.

6.3 The WebRTC Standard

This section presents a general introduction of the WebRTC architecture and how a WebRTC session is established between two peers. Moreover, the bandwidth estimation performed between remote WebRTC endpoints is also briefly discussed. It is worth stressing that the goal of this chapter is not to optimize the low-level components of WebRTC described in this section, but rather improving its video delivery architecture to guarantee better scalability, as detailed in Sections 6.4 and 6.5.

6.3.1 WebRTC Architecture and Session Initiation

WebRTC is a collection of communication protocols and APIs that enable real-time communication among remote peers[2] and is currently being standardized by the world wide web consortium and the Internet engineering task force. WebRTC is by design browser-based and does not rely on any external plug-in or other third-party software, which means that it is platform and device independent. Ideally, each device equipped with a browser is capable of initiating a WebRTC session, which makes it an ideal candidate for new interactive streaming applications, as telehealth or remote teaching. WebRTC is not only limited to video and audio calls, but can also be used for peer-to-peer file sharing and text messaging.

When two or more WebRTC peers want to establish a connection, three types of information have to be exchanged among them. First, session control messages have to be exchanged to initialize (or close) the communication and report errors. Second, network configuration messages are sent to communicate the IP addresses of the remote peers. In WebRTC in fact, the media, be it video, audio or text, is transferred in a peer-to-peer fashion between two or more participants. This aspect entails that the remote peers have to be aware of the respective IP addresses. Third, information related to the media are exchanged among the peers to find the media

[2]We refer to a general WebRTC protocol in this chapter by referring to all the functionalities and protocols that are included in WebRTC itself.

configuration that can be supported by all participants (e.g., codec and resolution in the case of video). This message exchange is called signaling and is not part of WebRTC itself, but relies on pre-existing protocols (e.g., SIP). All these message exchanges, which take place before the actual communication can start, are carried out with the help of an external server, called the signaling server, which supports the Session Description Protocol (SDP). SDP is used by the remote peers to inform the others about the transport protocol, ports, codecs and relevant parameters to be used during the media transfer.

As the media is exchanged directly among the participants, the protocol has been developed to cope well with firewalls and Network Address Translation (NAT). For this purpose, a Session Traversal Utilities for NAT (STUN) server is used. A STUN server allows NAT clients to find out their public address, the type of NAT they are behind and the port associated by the NAT with a particular port. In most cases, a STUN server is only used by the WebRTC peers during the connection setup, while the actual media is exchanged directly once the session is established. However, when direct media traffic is not allowed (e.g., because of a firewall), a Traversal Using Relays around NAT (TURN) server relays the messages and the media between two or more clients.

6.3.2 Congestion Control and Bandwidth Estimation in WebRTC

WebRTC uses UDP instead of TCP at the transport layer, as TCP cannot guarantee the low latency required in real-time communication, since its main focus is on reliability. In WebRTC, the congestion control mechanism is implemented at the application layer using the Real-Time Transport Protocol (RTP) and its control protocol RTCP. The congestion control mechanism tries to estimate the capacity of the channel connecting two remote WebRTC peers. This way, the sending peer can adjust the video encoding bitrate to the available bandwidth of the receiving peer. In contrast with classical video streaming techniques based on HTTP adaptive streaming, in WebRTC the sending peer directly controls the rate at which the video is sent.

The congestion control algorithm currently implemented in WebRTC is the Google Congestion Control (GCC) [23], which attempts to detect congestion before it actually occurs, by using the inter-arrival delay of consecutive packets. Particularly, the congestion control is divided in two separate parts: a delay-based controller located at the receiving peer, and a packet loss-based controller located at the sending peer. The receiver analyzes the inter-packet delay and generates an estimation of the available bandwidth. This report is sent back to the sender with an RTCP message called Receiver Estimated Maximum Bitrate (REMB), usually every 250 to 500 ms. To decide the final value of the encoding video bitrate, the

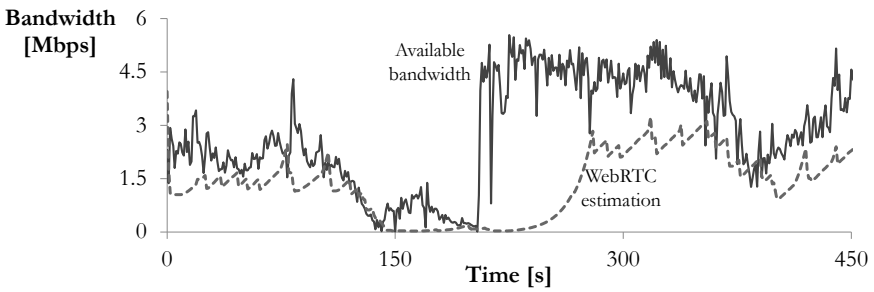


Figure 6.1: Illustrative example of the bandwidth estimation evolution in WebRTC. The WebRTC estimated bandwidth (dashed line) slowly follows the actual available bandwidth (full line), and is characterized by exponential increases and sudden decreases.

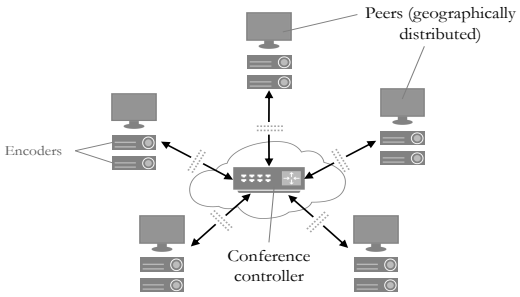
sender also uses the latest packet loss feedback, which is reported by the receiver in particular RTCP messages. The system is designed to slowly increase the estimated bandwidth and the video rate as long as no congestion is detected and to ensure that the available bandwidth of the channel is eventually matched. As soon as congestion is detected, the estimated bandwidth is decreased.

This approach gives a particular and characteristic evolution to the end-to-end estimated bandwidth in WebRTC. An example of such evolution is presented in Figure 6.1, which reports the result of an experiment with two WebRTC peers in communication among each other. In the experiment, a sending peer *A* is connected to a receiving peer *B* via a network link, whose available bandwidth is shaped as depicted in Figure 6.1 (full line). The actual bandwidth estimated by peer *A* (dashed line), which drives the encoding bitrate of the video sent to peer *B*, slowly follows the available bandwidth and results in an exponential increase followed by sudden drops when the bandwidth decreases.

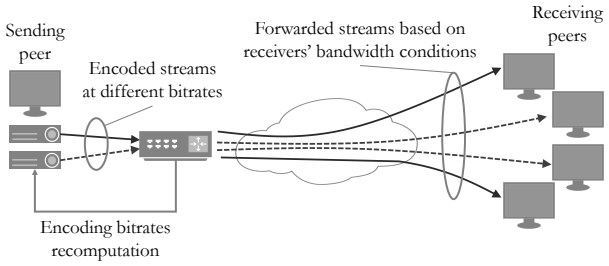
Congestion control algorithms in WebRTC are developed assuming that sending and receiving peers directly communicate and exchange media traffic among each other. In the proposed framework instead, this assumption is not true anymore, as the conference controller behaves as the endpoint for both the sender and the receivers. The impact on the bandwidth estimation of the receivers of this modification is discussed in Section 6.5.3.

6.4 Architecture of the Proposed WebRTC Framework

In this section, we describe the architecture of the proposed framework in a general remote collaboration scenario. The WebRTC framework presented in this chapter is proposed to optimize the delivery of real-time WebRTC streams in the context of



(a) The proposed framework is composed of several peers, each equipped with a number of encoders. The communication among the peers occurs through the conference controller, located between them (e.g., in the cloud). The peers are geographically distributed and can experience different bandwidth conditions.



(b) *From sender to receivers:* the conference controller behaves as the terminal endpoint for both the sender and the receivers, and performs the dynamic stream forwarding and dynamic bitrate recomputation tasks.

Figure 6.2: General architecture of the proposed WebRTC framework (6.2a), together with the high-level description of the tasks performed by the conference controller when optimizing the communication from the sending to the receiving peers (6.2b).

remote video collaboration applications. Particularly, a centralized node called the conference controller is used to guarantee cost-efficiency in terms of the number of encoders used at sender-side and scalability when the number of remote peers is large. Consequently, the framework does not aim to optimize the WebRTC protocol described in Section 6.3, but rather to improve its delivery architecture.

The high-level architecture of the proposed framework is presented in Figure 6.2a. In a general remote collaboration applications, several peers participate in the same remote session and are in communication among each other via the conference controller, which is positioned between them. Each peer is equipped with a limited number of encoders, usually much smaller than the number of participants. We assume that at any given point in time a set of peers is transmitting to a set of receiving peers. It is worth noting that, given the interactive nature of this communication, the role of sender and receiver is not static but can dynamically vary among the peers.

The goal of the controller is to optimize the delivery of the WebRTC streams from senders to receivers, as described in Figure 6.2b (for the sake of simplicity, the figure only depicts one sender). At sender-side, the remote event is captured and encoded in different streams at different bitrates, which are sent to the conference controller. The controller is then responsible for the dynamic forwarding of the encoded streams to all the receivers participating in the session, based on their bandwidth conditions. This approach allows to relieve the peer-to-peer architecture of classical WebRTC and improve the scalability of the system. Indeed, each encoder at sender-side is now associated with multiple receivers, with the number of receivers usually much larger than the number of encoders. To maximize the rate delivered to the receivers, the video bitrate of the encoders is not static, but is periodically recomputed by the controller to better follow the bandwidth conditions of the receivers. The functionalities carried out by the conference controller are detailed in Section 6.5.

The proposed framework can be applied in any remote collaboration scenario. In the remainder of this chapter though, we relax this condition and consider a one-to-many scenario only, typical in a virtual classroom, where one participant, called the *sender* is responsible for most of the communication towards the remaining peers, called *receivers*. In this case, the conference controller is located at the sender-side premises, so that enough bandwidth is always available between sender and controller. Interactivity is still envisioned in this case, as the receivers can communicate among each other or with the sender (e.g., asking questions in a virtual classroom). Particularly, we will focus on the downstream side of the problem (e.g., from sender to receivers), as most of the communication follows this path. The operations performed by the conference controller in this case are detailed in the next section. It is nevertheless implied that the proposed framework can guarantee the required level of interactivity, by allowing the receivers to participate in the communication.

6.5 WebRTC Conference Controller Design

The most important component of the proposed WebRTC framework is the conference controller, which performs two main tasks. First, the controller receives all the encoded streams from the sender and dynamically forwards them to the receivers, based on their available bandwidth (Figure 6.2b). Second, it periodically recomputes the encoding bitrates of the sender to better follow the long-term network variations of the receivers. In the remainder of this section, we detail the operations performed by the conference controller in terms of dynamic stream forwarding (Section 6.5.1) and encoding bitrate recomputation (Section 6.5.2). Moreover, we discuss in Section 6.5.3 the bandwidth probing mechanism employed by the conference controller to make a better estimation of the receivers' bandwidth.

6.5.1 Dynamic Stream Forwarding

The conference controller acts as an endpoint for the sender and for the receivers, by receiving the encoded streams from the sender and forwarding them to the receivers. Particularly, the controller forwards the highest sustainable stream to each receiver, based on their estimated bandwidth. As explained in Section 6.3, in a classic peer-to-peer WebRTC architecture, a receiver reports statistics and feedback to the corresponding sender, which allow to estimate the end-to-end bandwidth. In the proposed framework, the conference controller acts as the actual sender for the receivers. This aspect entails that the controller can intercept the REMB messages, which are used to estimate the bandwidth of the remote receivers [24]. Each time an REMB message is received by the controller, the corresponding bandwidth estimation is updated and a new stream is selected for the receiver, if needed. REMB messages are usually generated by the receivers every 250 to 500 ms. Consequently, the dynamic forwarding is performed at a very fine-grained timescale, which allows to accommodate the short-term bandwidth variations of the receivers and guarantee a continuous playback.

6.5.2 Encoding Bitrate Recomputation

A second, more long-term optimization is performed by the conference controller to recompute the set of encoding bitrates at sender-side. In the proposed framework, each encoder at sender-side transmits to multiple receivers at the same time. In order to maximize the video rate received by the receivers, the encoded rate should be as close as possible to the actual bandwidth of the receivers. Static, fixed encoding bitrates are suboptimal, as the receivers' conditions can change over time, especially in wireless environments, where the available bandwidth can highly fluctuate. By allowing the encoding bitrates to dynamically vary, it is possible to follow the long-term bandwidth variations of the receivers and, therefore, maximize the delivered video quality.

We formulate the dynamic bitrate recomputation problem as an ILP formulation, which is executed every T_{opt} seconds by the conference controller. At time t when the recomputation takes place, the virtual classroom is composed of R receivers, each associated with a bandwidth measure b_r. Different options are possible on how to compute b_r for a given time window of multiple seconds, as for example the average or the minimum estimated bandwidth over the period $[t - T_{opt}; t]$. In Section 6.7, we identify the best method to compute the bandwidth measure b_r from the individual bandwidth estimations of the receiver in the interval $[t - T_{opt}; t]$. The main sender of the virtual classroom is equipped with l_{max} encoders (with $l_{max} \ll R$), which can encode the video in the range $[B_{min}; B_{max}]$, where B_{min} and B_{max} are the minimum and maximum encoding rates, respectively. We indicate with L the number of possible encoding levels in

this interval, each associated to a rate B_l. The goal of the controller is to select the l_{max} encoding levels, among the possible L levels, which are the closest to the bandwidth measures b_r of the receivers. The complete ILP formulation is as follows:

$$
\begin{array}{lrcll}
\min\limits_{\alpha} & \sum\limits_{r=1}^{R}\sum\limits_{l=1}^{L} \alpha_{r,l}\,(b_r - B_l)^2 & & & \\
\text{s.t.} & \alpha_{r,l} & \in & \{0,1\} & \forall r \in \{1,\ldots,R\}, l \in \{1,\ldots,L\} \\
& \beta_l & \in & \{0,1\} & \forall l \in \{1,\ldots,L\} \\
& \beta_l & \geq & \alpha_{r,l} & \forall r \in \{1,\ldots,R\}, l \in \{1,\ldots,L\} \\
& \sum\limits_{l=1}^{L} \alpha_{r,l} & = & 1 & \forall r \in \{1,\ldots,R\} \\
& \sum\limits_{l=1}^{L} \beta_l & \leq & l_{max} & \\
& \beta_0 & = & B_{min} & \\
& \sum\limits_{l=1}^{L} \alpha_{r,l} B_l & \leq & b_r & \forall r \in \{1,\ldots,R\}
\end{array}
\tag{6.1}
$$

The solution of the problem is characterized by two sets of boolean decision variables, namely $\alpha_{r,l}$ and β_l. $\alpha_{r,l}$ is equal to 1 when client r is associated to encoding level l, and 0 otherwise. Similarly, β_l is equal to 1 when encoding level l is selected for one of the encoders, and 0 otherwise. The optimization problem is designed to find the l_{max} encoding levels whose bitrates allow to minimize the quadratic difference with the receivers' bandwidth measures. The first three constraints of the ILP formulation set up a consistent relation between the decision variables α and β. The fourth constraint indicates that each receiver can only be associated with one specific encoding level. The last three constraints are representative of the analyzed problem. First, only l_{max} encoding levels can be selected out of the L available levels (constraint 5), as l_{max} indicates the number of encoders available at sender-side. Second, we always select the lowest possible encoding bitrate as a solution (constraint 6). This way, we guarantee the receivers can always play the lowest available quality and avoid playout interruptions. Third, the encoding bitrate associated to receiver r must be lower than the bandwidth measure for r (constraint 7), in order to guarantee a continuous playout.

The ILP formulation presented in Equation 6.1 uses an objective, video-agnostic objective function in the video rate domain, namely the quadratic difference between the possible encoding levels and the bandwidth measures of the receivers. The objective function can be easily modified to perform the optimization in the Peak-Signal-to-Noise-Ratio (PSNR) domain instead, in order to select the l_{max} encoding levels that allow to maximize the actual video quality of the receivers, as shown in Equation 6.2:

$$\min_{\alpha} \quad \sum_{r=1}^{R} \sum_{l=1}^{L} \alpha_{r,l} \left(PSNR\left(b_r\right) - PSNR\left(B_l\right) \right)^2 \tag{6.2}$$

The constraints of this ILP formulation are the same as those presented in Equation 6.1. The main drawback of this approach is that the function mapping the video rate to the actual PSNR has to be estimated online by the conference controller. In Section 6.7, we show how the formulation in the PSNR domain can indeed be beneficial for the receivers' video quality, using a pre-computed PSNR curve.

Even though the presented ILP formulation can provide the optimal solution to the analyzed problem, it can suffer from scalability issues when the solution space, which depends on the number of receivers R and possible encoding levels L, is large. In order to be effective, the optimal solution should be computed in a fraction of the optimization period T_{opt}, which cannot be guaranteed when the solution space is too large. For this reason, we propose to use the K-means clustering algorithm to solve the bitrate recomputation problem in an approximate but highly scalable way. In this case, the inputs of the algorithm are the bandwidth measures b_r of the receivers, which have to be clustered into l_{max} separate clusters. Once the clusters are generated, the smallest value associated to each cluster is chosen as encoding bitrate for the l_{max} encoders. As for the ILP formulation, also in this case the lowest available encoding level is always selected, to guarantee a continuous playback. In Section 6.7, we evaluate both the ILP formulation and K-means algorithm in terms of system performance and computing time.

As a final consideration, it is worth noting that the two tasks carried out by the conference controller, namely dynamic forwarding and bitrate recomputation, are performed at different timescales. The bitrate computation presented in Equation 6.1 is executed on a timescale of seconds, and is used to adjust the encoding bitrates to take into account long-term variations of the receivers' network conditions. On the contrary, the stream forwarding is executed on a timescale of milliseconds, to closely follow the changing bandwidth conditions of the receivers.

6.5.3 Bandwidth Probing

Another task carried out by the conference controller is the estimation of the receivers' bandwidth, which can be carried out using REMB messages (see Section 6.5.1) and probing. As explained in Section 6.3, in a standard peer-to-peer WebRTC architecture, the sender decides the encoding bitrate of the video sent to the receiver. The encoding bitrate is decided based on the available bandwidth towards the receiver, which is estimated using a congestion control algorithm. The encoding rate is slowly increased to detect the capacity of the channel. When the maximum channel capacity is about to be reached, congestion is detected by the

receiver, which communicates this feedback to the sender. This mechanism allows to discover the actual capacity of the channel connecting the sender to the receiver, independently of the actual implemented congestion control algorithm. This process assumes that the sender and the receiver are in direct communication between each other, so that each perturbation of the encoded bitrate at the sender is reflected in the congestion experienced by the receiver.

Unfortunately, this assumption does not hold anymore when the conference controller is introduced. Indeed, the controller acts as an endpoint for both the sender and the receivers, which are no longer in direct communication with each other. Moreover, the encoding bitrates of the video are decided by the controller itself, which computes them solving the optimization problem presented in the previous section. In standard WebRTC, the video traffic is used by the endpoints to estimate the capacity of the channel, while this condition is not valid anymore when the conference controller is introduced. To relieve this issue, the conference controller can send additional *dummy* probing traffic towards the receivers, to improve the channel capacity estimation. The probing mechanism has to be designed to balance two opposite objectives. First, the probing traffic should not be too aggressive to avoid congesting the video itself. Second, it should be responsive enough to detect the actual channel capacity of the receivers. Each receiver in the remote session is associated with a separate and independent probing instance at the conference controller. This aspect allows to follow the bandwidth variations of each receiver, independently of each other.

In the remainder of this section, we detail the probing mechanism developed for the conference controller. It is worth mentioning that, due to the continuous ongoing development of the WebRTC standard, the proposed approach represents an initial attempt for bandwidth probing at an SFU, whose performance and limitations are discussed in Section 6.7.2. As an example, the Jitsi-Videobridge software already provides low-level probing functionalities[3]. Moreover, probing can be considered as an additional tool used by the conference controller. The main tasks performed by the controller, namely stream forwarding and bitrate recomputation, are completely independent from the implemented probing mechanism, and can be executed even if probing is not enabled or is not needed.

The amount of probing traffic $P(-)$ to be sent towards the receivers is recomputed every K_{prob} seconds by the conference controller. Particularly, at time instant k, an initial, tentative amount of probing traffic, indicated with $p(k)$, is computed as in Equation 6.3:

$$p(k) = \begin{cases} \lambda(k) \times vr(k) & \text{if } bw(k) < bw(k - K_{prob}) \\ (1 + \lambda(k)) \times P(k - K_{prob}) & \text{otherwise} \end{cases} \tag{6.3}$$

[3]https://github.com/jitsi/jitsi-videobridge/blob/master/src/main/java/org/jitsi/videobridge/cc/BandwidthProbing.java

$vr(k)$ is the rate of the video sent to the receiver at time k, $bw(k)$ is the estimated bandwidth and $P(k - K_{prob})$ is the probing traffic sent at time $k - K_{prob}$. When the estimated bandwidth of the receiver drops (first condition in Equation 6.3), the probing traffic is reset to a fraction of the video rate sent to the receiver. This sudden decrease avoids that the probing traffic congests the channel and impairs the actual video stream. Otherwise, the probing traffic is tentatively increased (second condition in Equation 6.3), following a multiplicative increase driven by parameter $\lambda(k)$. This parameter is not fixed to a static value, but depends on the video rate $vr(k)$. Particularly, $\lambda(k)$ is higher when $vr(k)$ is lower, and vice-versa. When the video rate is low, more probing traffic is needed to detect the channel capacity. Consequently, λ becomes larger to make probing more aggressive. On the contrary, λ decreases when vr is high, thus making probing more conservative and avoiding congesting the channel. The evolution of the λ parameter is presented in Equation 6.4:

$$\lambda(k) = c + m \times vr(k) \qquad \text{with} \qquad m = -\frac{\lambda_{max}}{B_{max}}, \quad c = \lambda_{max} - m \times B_{min} \tag{6.4}$$

where B_{min} and B_{max} are the minimum and maximum encoding bitrates, and λ_{max} is the highest possible value for the λ parameter. $\lambda(k)$ decreases linearly as the video rate $vr(k)$ increases, ranging between λ_{max} and $\lambda_{max} \times (B_{min}/B_{max})$, when $vr(k)$ is equal to B_{min} and B_{max}, respectively. As a final step, the actual amount of probing traffic $P(k)$ is limited in case it could congest the channel in the next interval $[k; k + K_{prob}]$:

$$P(k) = \begin{cases} \eta \times (bw(k) - vr(k)) & \text{if } p(k) + vr(k) \geq bw(k) \\ p(k) & \text{otherwise} \end{cases} \tag{6.5}$$

When the total amount of data sent to the receiver (i.e., $p(k) + vr(k)$) is higher than the current estimated bandwidth $bw(k)$, probing should be limited in order to avoiding congesting the channel. Therefore, the actual final probing $P(k)$ is set to a fraction of the remaining estimated channel capacity $bw(k) - vr(k)$, via the parameter η. A small η results in a more conservative but slower probing behavior, while a large value could result in congestion that could impair the video stream itself. Otherwise, the amount of probing traffic is simply set to the value $p(k)$ computed in Equation 6.3.

6.6 Implementation Details

In order to evaluate the performance of the proposed solution in a realistic environment, the framework is implemented on the imec iLab.t Virtual Wall emula-

tion platform[4]. To implement the receivers and the sender, the Google Chrome browser is used. Nothing is changed of the Google's original WebRTC stack, which makes our solution completely WebRTC-compliant. From an implementation perspective, the sender is decoupled into l_{max} WebRTC sub-senders, each encoding the video stream at a different bitrate. To implement the conference controller, the Jitsi software[5] is used, a set of open-source projects to build and deploy secure videoconferencing solutions. Particularly, the Jitsi-Videobridge[6], a WebRTC SFU, has been used as the main component. Its default functionality is to relay all the streams generated by the conference to all the participants. Jitsi-Meet[7], a JavaScript application running on top of the browser WebRTC stack, is used at the sub-senders and receivers to interface with the Jitsi-Videobridge. In the remaining of this section, we explain the modifications we made on the Jitsi-Videobridge to implement the stream forwarding and bitrate recomputation functionalities.

6.6.1 Stream Forwarding Selection

By default, the Jitsi-Videobridge forwards multiple streams to a participant. The stream of the participant who is currently speaking, the so-called *dominant speaker*, is automatically detected by the software and is always included in these streams. We override this logic so that a different dominant speaker can be manually set per receiver. Moreover, we limit the amount of streams that can be sent to a specific receiver to only one, selected as previously explained. Using this mechanism, the Jitsi-Videobridge dynamically assigns a sub-sender per receiver, so that the encoding bitrate is lower than the receiver's estimated bandwidth.

To implement this functionality, the conference controller has to estimate the available bandwidth of the receivers first. This estimation is performed by default by the Jitsi-Videobridge, which forwards all WebRTC traffic among the conference participants, implemented using the RTP/RTCP protocol suite. In WebRTC, bandwidth estimation is performed using RTCP REMB messages, a feedback used by a receiver to notify its media stream sender over the same RTP session of the estimated available bandwidth on the path to the receiving side. The Jitsi-Videobridge intercepts these RTCP REMB messages and is therefore aware of the available bandwidth of the receivers.

6.6.2 Dynamic Encoding Bitrate Recomputation

As the long-term network conditions of the receivers can change over time, it is required to periodically recompute the set of encoding bitrates of the sub-senders,

[4] http://doc.ilabt.iminds.be/ilabt-documentation/virtualwallfacility.html
[5] https://jitsi.org/
[6] https://github.com/jitsi/jitsi-videobridge
[7] https://github.com/jitsi/jitsi-meet

as described in Section 6.5. Once these values are computed, they have to be enforced on the WebRTC sub-senders. However, there is no standardized way to set the encoding bitrate of a WebRTC client. To perform this task, we use the RTCP REMB messages, which contain the receiver's estimated available bandwidth. In WebRTC, the congestion control mechanism of a sender considers this estimation as the maximum bitrate that can be sent to a receiver. Consequently, the sender's encoder uses this value as its current target bitrate. Once the new bitrates are computed, the Jitsi-Videobridge modifies the REMB feedback messages for the sub-senders by setting the newly computed bitrate instead of the bandwidth estimation of the receivers. This way, the sub-senders are forced to modify their encoding bitrates. To implement this mechanism, we changed the way RTCP messages are generated in libjitsi[8], the underlying Java media library used by Jitsi-Videobridge. Instead of setting the maximum bitrates in the REMB messages for the sub-senders to the latest estimated remote bandwidth, we set them to the bitrates generated by the bitrate recomputation presented in Section 6.5.

6.7 Performance Evaluation Results

In this section, we perform a detailed analysis of the performance of the proposed framework. In Section 6.7.1, we first present a Java-based WebRTC simulator that allows to thoroughly test the theoretical performance of the system under a large number of configurations in terms of number of receivers, maximum number of encoders, bitrate recomputation period and employed algorithm (ILP or clustering). In Section 6.7.2, we then use the emulation testbed presented in the previous section to confirm the simulation results in a realistic environment and to highlight the main differences between simulation and emulation.

6.7.1 Simulation Results

The Java-based simulator presented in this section has been developed to mainly test the performance of the bitrate recomputation presented in Section 6.5.2. Consequently, only the functionalities of the WebRTC receivers and the conference controller are implemented in the simulator, while no actual video encoding, dynamic forwarding or bandwidth probing is performed. This simplification allows us to isolate the gains of the proposed bitrate recomputation. A more realistic evaluation is instead performed in the next section using the emulation setup presented in Section 6.6. The role of the simulated WebRTC receivers is to provide input on the bandwidth measurements that are periodically fed to the simulated conference controller, which performs the bitrate recomputation. The ILP formulation pre-

[8]https://github.com/jitsi/libjitsi

Algorithm 1: WebRTC-like bandwidth evolution used in the Java-based simulator.

Require: t, current simulation time, in seconds
$ban(t)$, actual available bandwidth for the simulated WebRTC receiver at time t, in kbps
$webRTCBan(t-1)$, WebRTC-like receiver bandwidth estimation at the previous time step $t-1$, in kbps
Ensure: $webRTCBan(t)$, WebRTC-like receiver bandwidth estimation at time t, in kbps

1: **if** $t - dropTime \leq 15$ **then**
2: $webRTCBan(t) = (1 + 0.016) \times webRTCBan(t-1)$
3: **else**
4: $webRTCBan(t) = (1 + 0.075) \times webRTCBan(t-1)$
5: **end if**
6: **if** $webRTCBan(t) > ban(t)$ **then**
7: $webRTCBan(t) = ban(t)$
8: $dropTime = t$
9: **end if**
10: **return** $\max(webRTCBan(t), 30)$

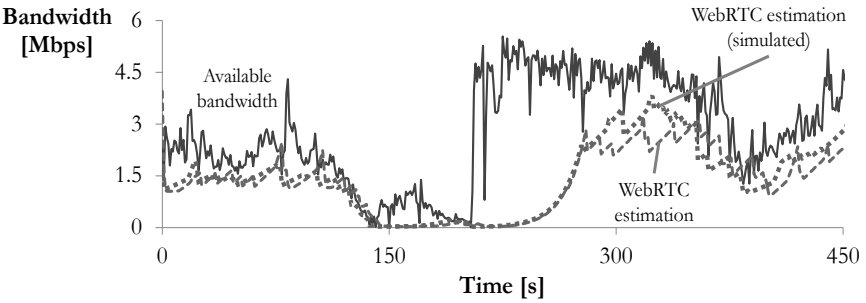


Figure 6.3: WebRTC bandwidth estimation (dashed line) slowly follows the available bandwidth (full line). The WebRTC-like bandwidth evolution presented in Algorithm 1 (dotted line) closely follows the real WebRTC bandwidth.

sented in Section 6.5.2 is solved using the CPLEX software[9], while the K-means algorithm has been implemented using the Weka library [25].

In order to generate a realistic behavior of the WebRTC receivers, we empirically model the characteristic pattern of the bandwidth estimated by the WebRTC receiver, discussed in Section 6.3 and shown in Figure 6.1. This modeling, presented in Algorithm 1, allows to obtain a WebRTC-like bandwidth evolution in the Java-based simulator. Given the actual available bandwidth at time t, the WebRTC bandwidth evolves following a multiplicative increase (lines 2 and 4). The multiplicative factor depends on the time elapsed since the last detected bandwidth drop. Particularly, the bandwidth tends to increase slowly during the few seconds following the drop (line 1), and faster afterwards. Moreover, the WebRTC simulated bandwidth cannot drop below 30 kbps (line 10), which is the minimum value

[9]https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Table 6.1: Overview of the evaluated parameter configuration in the Java-based simulator, resulting in a total of 1890 different configurations.

Parameter	*Value*
$[B_{min}; B_{max}]$	[50;2500] kbps
L	40
l_{max}	2, 3, 4, 5, 6, 8, 10
R	5, 10, 20, 40, 80
Bitrate recomputation	Static, ILP, K-means
Optimization domain	Rate, PSNR
T_{opt}	2, 4, 8, 16 s
b_r *computation method*	Latest, Minimum, Average

returned by the bandwidth estimator implemented by libjitsi[10]. Figure 6.3 reports the evolution of the simulated WebRTC bandwidth presented in Algorithm 1, for the same illustrative bandwidth pattern presented in Figure 6.1. The simulated bandwidth closely resembles the actual estimated bandwidth. It is worth stressing that this algorithm is only used in the simulator to obtain realistic values for the bandwidth estimations of the WebRTC receivers and is not intended to provide an exact modeling of the WebRTC bandwidth estimation algorithm. The constant values reported in Algorithm 1 have been set after fine tuning the algorithm on a set of bandwidth traces collected on a real 3G network [26].

Using the described simulator, a large-scale evaluation of the parameters of the bitrate recomputation problem is carried out, which is presented in Table 6.1. In total, 1890 different configurations have been evaluated. To analyze the approach under realistic network conditions, we apply a different bandwidth pattern, collected on a real 3G network [26], on each simulated WebRTC receiver. The available bandwidth for one client fluctuates between 202 bps and 6335 kbps, with an average of 2087 kbps and a standard deviation of 1314 kbps. Each experiment configuration has been repeated fifteen times to guarantee statistical significance, each iteration lasting 240 seconds. The minimum (B_{min}) and maximum (B_{max}) encoding rates are fixed to 50 and 2500 kbps. Forty possible encoding levels (L) are available, with bitrates equally spaced in the encoding rate interval. We tested several values for the number of encoders at sender-side (l_{max}), ranging from 2 to 10, and number of receivers (R), from 5 to 80. This choice allows to test the framework both when $R \simeq l_{max}$ and when $R \gg l_{max}$. We compare the performance of the proposed framework to a solution where the encoding bitrates are statically fixed and are not dynamically recomputed during the experiment. When the optimization takes place in the video rate domain, the static bitrates are equidistantly assigned to $50 + l \times (2500 - 50)/(l_{max} - 1)$ kbps, with $l = 0, 1, \ldots, l_{max} - 1$. In the PSNR domain, the l_{max} static bitrates are assigned to be equidistant in terms

[10]https://github.com/jitsi/libjitsi/blob/master/src/org/jitsi/impl/neomedia/rtp/sendsidebandwidthestimation/BandwidthEstimatorImpl.java

of PSNR values in the interval [50; 2500] kbps. We generated a PSNR-like curve to be used in the simulator, which follows a logarithmic function of the video rate, as proposed by Schroeder et al. [27]:

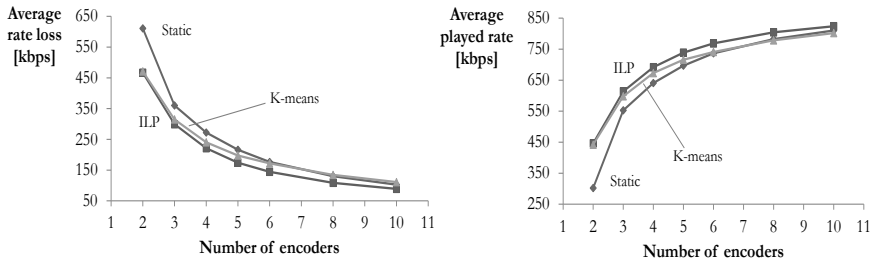
$$PSNR = 3.136 \times log(rate) + 18.297 \tag{6.6}$$

The generated PSNR values range between 30.56 dB and 42.77 dB for a video rate of 50 kbps and 2500 kbps, respectively. Even though the generated PSNR curve is artificial, it nevertheless allows us to show how the proposed bitrate recomputation changes when carrying out the optimization in the video rate or PSNR domain. Changing the curve would clearly change the absolute results, but the general conclusions would still be valid nonetheless. In the ILP and K-means cases, the recomputation period (T_{opt}) ranges from 2 to 16 seconds. A shorter period is desirable, as it allows to closely follow the network conditions of the receivers, but it is computationally expensive when the solution space (defined by the number of receivers and possible encoding levels) is large. We also tested three different methods to compute the bandwidth measure b_r used in the optimization problem in Section 6.5.2. Assuming the optimization takes place at time t and BW indicates the vector containing all the N receiver bandwidth estimations in the interval $[t - T_{opt}; t]$, b_r is calculated as follows:

$$\text{Minimum: } b_r = \min(BW) \qquad \text{Average: } b_r = \frac{1}{N} \times \sum_i BW(i)$$
$$\text{Latest: } b_r = BW(N) \tag{6.7}$$

representing the minimum bandwidth estimation in the period $[t - T_{opt}; t]$, the average bandwidth estimation or the latest one, respectively.

For each receiver, we keep track of three metrics. First, the average video rate received during the experiment. Second, the average *rate loss*, computed as the difference between the available bandwidth at the receiver and the actual received rate. The rate loss represents the gap between the rate a receiver is able to sustain (i.e., the available bandwidth) and the rate actually received. Third, the average PSNR value achieved by the receiver, using the above mentioned equation. It is worth noting that the first two metrics are objective and video-independent. On the other hand, the PSNR results depends on the employed bitrate to PSNR curve, which is modeled as in Equation 6.6. To investigate the impact of the parameters presented in Table 6.1, we compute for each of the 1890 experiment configurations the average rate loss, played rate and PSNR over the whole group of clients and over the fifteen different bandwidth configurations. When analyzing the impact of one of the parameters listed in Table 6.1 on the performance of the system, we keep the remaining parameters fixed to reference values that we consider realistic for the virtual classroom scenario. Particularly, we consider as reference scenario



(a) Rate loss.

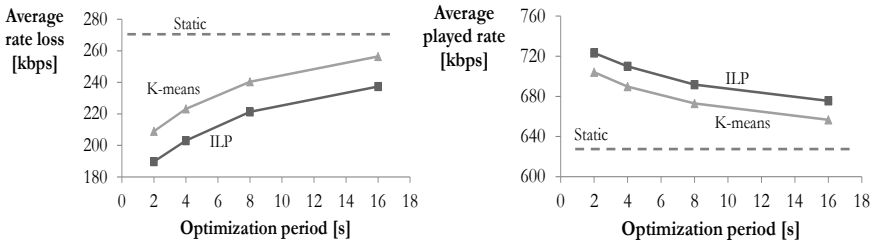
(b) Played rate.

Figure 6.4: Increasing the number of available encoders reduces the rate loss (6.4a) and increases the played rate (6.4b). The proposed approach outperforms a static one, and allows to achieve similar performance using less encoders.

the case with $l_{max} = 4$, $R = 20$, $T_{opt} = 8$s and b_r computed as the latest available bandwidth estimation. This approach is used to better highlight the impact of each single parameter on the proposed framework. Nonetheless, very similar conclusions can be drawn when altering the reference scenario.

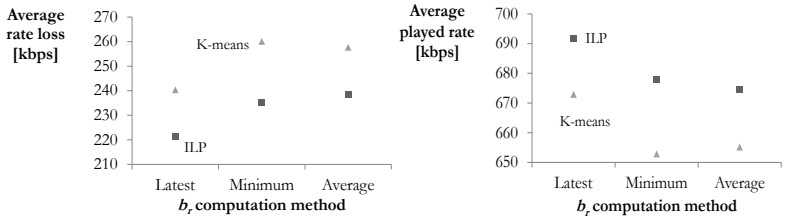
6.7.1.1 Optimization in the Video Rate Domain

In this section, we report the results when the optimization carried out by the conference controller takes place in the video rate domain (problem formulation presented in Equation 6.1). Figure 6.4 reports the average rate loss and played rate as a function of the number of available encoders at sender-side. Increasing the number of encoders reduces the rate loss and increases the played rate, independently of the used bitrate recomputation algorithm. When more encoders are used, it is possible to follow in a more fine-grained way the bandwidth variations of the receivers. Ideally, when each receiver is associated to a dedicated encoder, as in classic peer-to-peer WebRTC, the rate loss would drop to zero and the played rate would be maximized. The optimal ILP formulation, solved using the CPLEX software, clearly provides the best results. The gains are particularly evident when few encoders are used. When three encoders are available, the proposed dynamic bitrate recomputation results in 17% reduced rate loss and 11% increased played rate, compared to a static approach. Moreover, our solution is more efficient in terms of the number of encoders, as similar performance can be reached using fewer encoders. For instance, a similar rate loss and played rate is obtained in the static case using four encoders as in the dynamic case using three encoders. The K-means clustering approach reaches similar results, while being more scalable than the ILP formulation. In terms of rate loss, the K-means approach is less than 10% worse than the optimal formulation, when the number of encoders is small. When



(a) Rate loss. (b) Played rate.

Figure 6.5: Increasing the bitrate recomputation period has a negative effect on the performance of the system, as it is more difficult to follow the bandwidth variations of the receivers. Nevertheless, a longer optimization period reduces the computational complexity of the bitrate recomputation.



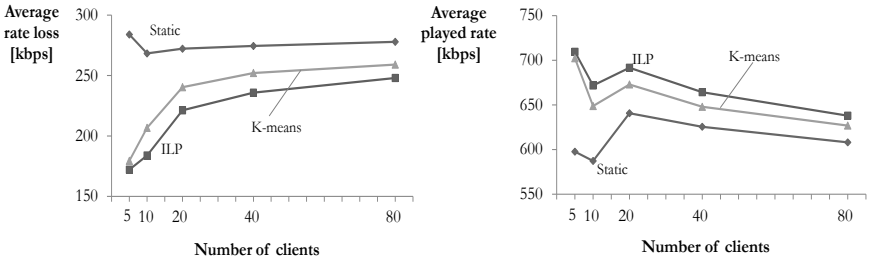
(a) Rate loss. (b) Played rate.

Figure 6.6: The best performance is reached when the bandwidth measure b_r, the input of the bitrate recomputation problem presented in Section 6.5.2, is computed as the latest bandwidth estimation of the receivers in the period $[t - T_{opt}; t]$.

the number of encoders increases, the sub-optimal clustering algorithm tends to reach similar performance as the static case.

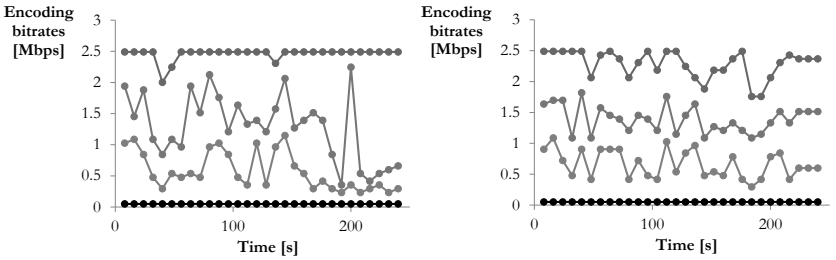
Another important parameter of the conference controller is the bitrate recomputation period, whose impact is shown in Figure 6.5. As expected, a longer optimization period degrades the performance of the system, both when the optimal ILP formulation and the clustering approach is used. The relevance of the encoding bitrates tends to decrease over time, as the long-term bandwidth conditions of the receivers might change. A shorter period can reduce this side effect, but it is more computationally expensive, especially when the number of receivers is large.

The ILP formulation and clustering algorithm presented in Section 6.5.2 take as inputs a general bandwidth measure b_r for each receiver. This bandwidth measure is a function of the bandwidth estimations of a particular receiver in the period $[t - T_{opt}; t]$ and can be computed as described in Equation 6.7. Particularly, we compute b_r as the minimum or average bandwidth estimation in the period



(a) Rate loss. (b) Played rate.

Figure 6.7: The relative gain of the proposed approach compared to a static one tends to decrease as the number of receivers increases.

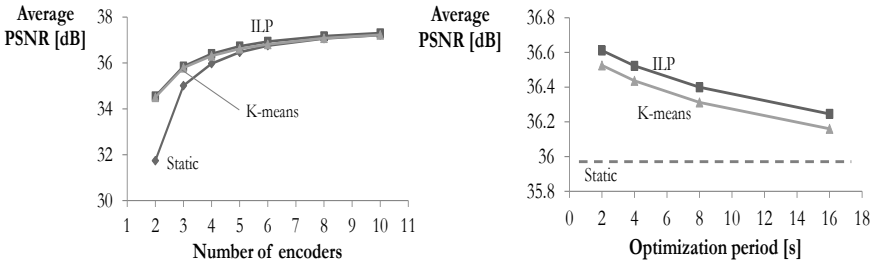


(a) Encoding bitrates – 10 clients. (b) Encoding bitrates – 80 clients.

Figure 6.8: As the number of receivers increases, the dynamic recomputation results in more stable and less variable encoding bitrates.

$[t - T_{opt}; t]$ or as the latest available bandwidth estimation. Figure 6.6 reports the rate loss and played rate, for both the ILP formulation and K-means algorithm, when the different bandwidth measure methods are used. In both the ILP and K-means cases, using the latest available bandwidth estimation as input for the bitrate recomputation yields the best results. The bandwidth estimation of WebRTC is not instantaneous but takes into account previous estimations as well. Moreover, the bandwidth evolution is rather slow in order to avoid congesting the video channel. This behavior is clearly visible in the exponential evolution captured by Algorithm 1 and visible in Figure 6.3. When b_r is computed as the average bandwidth estimation over the entire optimization period, old, non-relevant values are also included. Conversely, the latest bandwidth estimation best represents the actual bandwidth conditions of the receivers.

To conclude this analysis, we investigate whether the number of receivers has any influence on the performance of the system (Figure 6.7). Interestingly, the relative gain of the dynamic bitrate recomputation decreases with the number of



(a) Influence of the number of encoders.

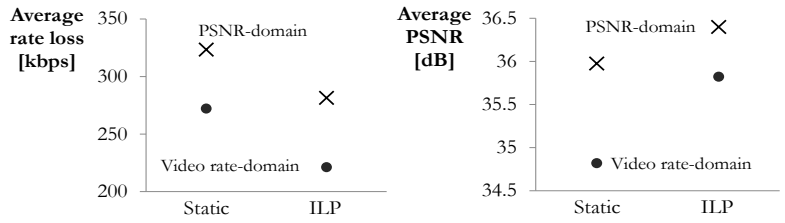
(b) Influence of the optimization period.

Figure 6.9: Evolution of the PSNR as a function of the number of encoders (6.9a) and the optimization period (6.9b).

receivers. For ten receivers, the rate loss is decreased by 31% when using the ILP formulation. This value decreases to 10% when eighty receivers are simulated. When the number of receivers is large, it becomes more difficult to recompute the bitrates in order to follow the bandwidth evolution of each individual receiver. As an example, Figure 6.8 reports the encoding bitrates evolution over time, for ten and eighty clients, with four available encoders. In the eighty receivers case, the recomputed bitrates are more stable than in the ten receivers case. In other words, when the number of receivers is large, the encoding bitrates tend to change less, resulting in a behavior that is more similar to a static association. In terms of scalability, even in the 80 receivers case, the optimal solution using the ILP formulation can always be found in less than 100 ms, when the experiments are carried out on a 2x Hexacore Intel E5645 (2.4GHz) CPU machine with 24GB RAM, running Ubuntu 16.04. A more in-depth analysis on the scalability of the proposed approach is reported in Section 6.7.1.4.

6.7.1.2 Optimization in the PSNR Domain

In this section, we briefly present the results when the bitrate recomputation is carried out in the PSNR domain (Equations 6.2 and 6.6). As for the optimization in the video rate domain, we consider as reference scenario the case with $l_{max} = 4$, $R = 20$, $T_{opt} = 8$s and b_r computed as the latest available bandwidth estimation. Figure 6.9a shows the evolution of the PSNR as a function of the number of encoders. Also in this case, increasing the number of encoders allows to increase the achieved PSNR. Compared to Figure 6.4b, the gain of the proposed approach is higher when few encoders are used, but decreases faster as the number of encoders increases. The optimization period plays a similar role as in Section 6.7.1.1. A longer optimization period results in poorer performance, as the encoding bitrates might not be representative anymore of the varying bandwidth conditions of the receivers. Similar results as those reported in the previous section are obtained for



(a) Rate loss. (b) PSNR.

Figure 6.10: The rate loss increases when the optimization is carried out in the PSNR domain. Similarly, the PSNR decreases when the optimization takes place in the video rate domain.

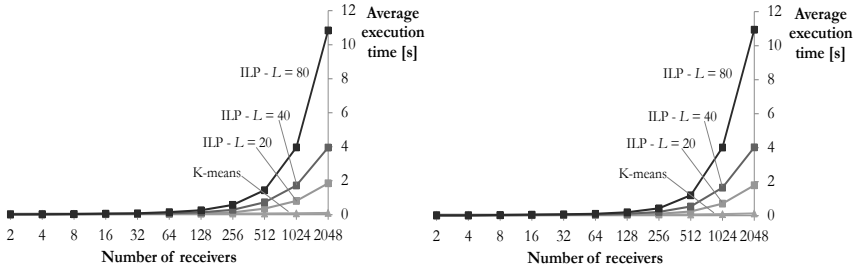
the b_r computation methods and the influence of the number of clients. For space reasons, these results are therefore omitted.

6.7.1.3 Comparison Between Video Rate and PSNR Domain Optimization

The goal of this section is to highlight the difference in performance arising when the bitrate recomputation is carried out in the video rate or PSNR domain. For this reason, we selected a particular configuration with twenty receivers and four available encoders, and report the average rate loss and average PSNR obtained using a static approach and the ILP formulation. In this last case, the optimization period is fixed to eight seconds and b_r is computed as the latest available bandwidth estimations of the receivers. Figure 6.10a reports the rate loss for the static and ILP approaches. When the optimization is carried out in the video rate domain, the goal is to reduce the rate loss itself (see objective function in Equation 6.1). Therefore, the rate loss is lower compared to the optimization in the PSNR domain, for both approaches. A similar conclusion can be drawn for the PSNR. The improvement in terms of PSNR in the static case clearly shows how the bitrate allocation changes when the objective is to maximize the video quality itself, rather than the achieved video rate. Even though the rate loss increases by 18%, the PSNR itself increases by more than 1 dB. It is worth stressing though that the PSNR results depend on the streamed video, while the rate loss is an independent and objective metric. Moreover, we assume in this chapter that the PSNR curve is pre-computed and available at the conference controller. In reality, the PSNR curve is not available and has to be estimated online by the conference controller.

6.7.1.4 Scalability Analysis

Besides being able to provide good performance in terms of rate loss and played rate, the proposed recomputation algorithms have to be able to compute the new



(a) Three encoders.

(b) Twelve encoders.

Figure 6.11: The proposed ILP formulation can scale well up to 128 receivers. On the contrary, the K-means algorithm can always provide a solution in less than 130 ms, even when the number of receivers is high.

set of encoding bitrates in a fraction of the optimization period. This requirement is necessary to guarantee that the new set of bitrates is still representative of the receivers' bandwidth conditions. In order to test the scalability of the ILP formulation and K-means algorithm, we perform a series of experiments with increasing number of clients, from 2 to 2048, and number of encoders, equal to 3 and 12. In the ILP case, we also test the influence of the parameter L, which indicates the number of possible encoding levels in the $[B_{min}; B_{max}]$ interval (see Section 6.5.2 and Table 6.1). The solution space of the ILP formulation depends on the number of receivers R and the number of possible encoding levels L, among which the l_{max} final encoding bitrates are selected. The results of this analysis are presented in Figure 6.11. All the experiments are carried out on a 2x Hexacore Intel E5645 (2.4GHz) CPU machine with 24GB RAM, running Ubuntu 16.04. Independently of the number of encoders and the number of possible encoding levels L, the ILP formulation can scale well up to 128 clients, providing a solution in less than 260 ms in the worst case scenario. This aspect entails that in a virtual classroom scenario, where the number of students is usually below 100, an optimal solution can always be computed. Depending on the value of L, more or less receivers can be supported using the ILP formulation. When $L = 20$ for instance, the ILP can still scale in the 512 clients case. Nevertheless, reducing this value too much can negatively affect the performance of the system. The number of possible encoding levels should indeed provide a fine-grained discretization of the encoding interval $[B_{min}; B_{max}]$, among which the ILP formulation can choose the encoding bitrates. The K-means clustering algorithm is instead able to scale extremely well, even for 2048 receivers, and provide the new encoding bitrates in less than 130 ms, in all cases. In light of the above, the ILP formulation can be used in nominal conditions, when the number of receivers is small, while the K-means algorithm can be employed when a large crowd of students attends the remote teaching session.

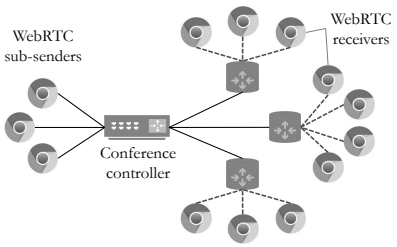


Figure 6.12: The emulated setup on the imec iLab.t Virtual Wall is composed of several WebRTC sub-senders and receivers implemented using the Google Chrome browser and one conference controller implemented using the Jitsi-Videobridge (Section 6.6).

Table 6.2: Overview of evaluated parameter configuration in the emulation testbed, resulting in a total of 24 different configurations.

Parameter	*Value*
$[B_{min}; B_{max}]$	[50;2500] kbps
L	40
l_{max}	3, 4, 5
R	10, 28
Bitrate recomputation	Static, ILP
Optimization domain	Rate
T_{opt}	4, 8, 16 s
b_r *computation method*	Latest

6.7.2 Emulation Results

In this section, we investigate the performance of the proposed framework in a realistic virtual classroom scenario, using the emulation testbed presented in Section 6.6, which allows to test all the functionalities of the conference controller (namely dynamic forwarding, bitrate recomputation and probing) with real software components. In the testbed, each WebRTC receiver is a separate physical machine running a Google Chrome browser, connected to the conference controller via a set of routers (see Figure 6.12). A different bandwidth trace is applied on the red dashed links connecting the routers to the receivers, while the black full links are over-provisioned. The single sender is split into several WebRTC sub-senders directly connected to the controller.

As in Section 6.7.1, we evaluate the system under different configuration settings, summarized in Table 6.2. The number of encoder (l_{max}) varies between 3 and 5, while the number of receivers (R) is equal to 10 or 28, which represent a typical virtual classroom scenario. Besides the static association, only the ILP formulation is tested. As shown in Section 6.7.1.4, an optimal solution can be found in less than 260 ms when the number of clients is less than 128. Therefore, we only present results for the optimal recomputation. The optimization takes place

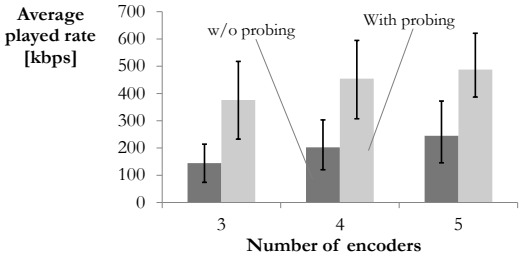


Figure 6.13: In emulation, probing allows to more than double the played rate, compared to a scenario where probing is disabled.

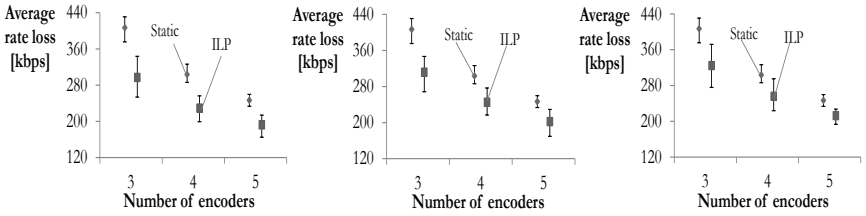
in the video rate domain only. The other parameters are similar to those presented in Table 6.1, to allow for an easy comparison between simulation and emulation. The latest available bandwidth estimation has been chosen as computation method for b_r. In Section 6.7.1.1, we showed that this choice leads to the best results. For consistency, the same 3G bandwidth traces used in the simulation experiments are also used in the emulation testbed. This aspect entails that the simulated and emulated WebRTC receivers are applied with the same bandwidth patterns, to guarantee consistency. Each experiment configuration has been repeated fifteen times to guarantee statistical significance, each iteration lasting 240 seconds. No real video capture is carried out by the encoders. Instead, a predefined video was used as input for the encoders[11].

For each emulated iteration, we compute the average rate loss and played rate over the entire group of receivers, and present the average results over the fifteen emulated iterations. Each point of the graphs is also associated with the 10% and 90% quantiles over the fifteen iterations.

6.7.2.1 Bandwidth Probing Effect

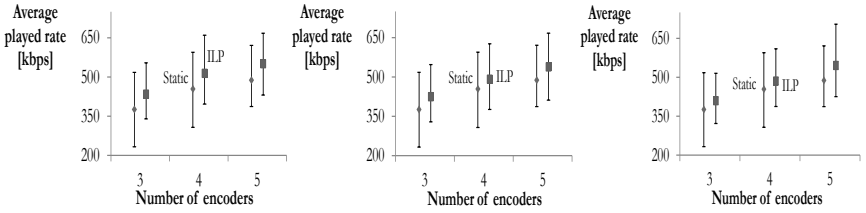
We start our analysis by first investigating the impact of the bandwidth probing algorithm, presented in Section 6.5.3, on the performance of the proposed framework. We select a configuration with 28 receivers and the static encoding bitrate allocation. This choice allows to isolate the benefits of probing as opposed to those due to the dynamic recomputation. The number of encoders varies between 3 and 5. When probing is enabled, the parameters λ_{max} and η (see Section 6.5.3) are fixed to 0.4 and 0.875, respectively. Preliminary results showed that these values provide the best probing performance. The results of this analysis are presented in Figure 6.13. When probing is enabled, the average played rate increases by 2.6 and 1.9 times in the three and five encoders case, respectively. When probing is disabled, the receivers and the conference controller, which acts as an end-point for

[11] https://media.xiph.org/video/derf/y4m/factory_1080p30.y4m



(a) $T_{opt} = 4$ s. (b) $T_{opt} = 8$ s. (c) $T_{opt} = 16$ s.

Figure 6.14: Impact of the optimization on the rate loss of the different bitrate recomputation approaches. The ILP formulation provides the best results. The difference between a dynamic and static association decreases when the optimization period increases.



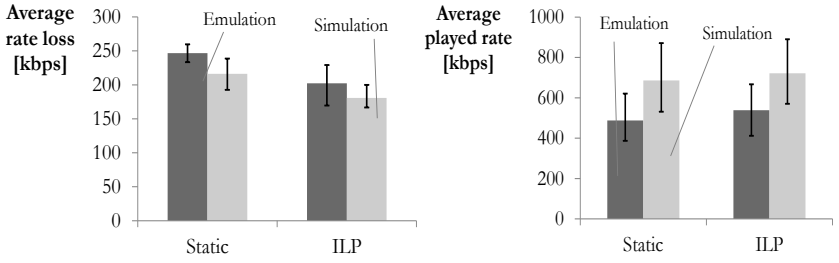
(a) $T_{opt} = 4$ s. (b) $T_{opt} = 8$ s. (c) $T_{opt} = 16$ s.

Figure 6.15: Impact of the optimization period on the played rate on the different bitrate recomputation approaches. As expected, the rate decreases as the optimization period increases, with the ILP formulation reaching the best performance.

the receivers in our framework, can only rely on the video traffic itself to perform the bandwidth estimation, which is not sufficient to correctly estimate the capacity of the channel. For this reason, the controller has to directly perform the bandwidth probing task that in a standard WebRTC architecture would be performed by the senders themselves. The bandwidth probing procedure presented in Section 6.5.3 can effectively improve the estimation of the receivers' bandwidth, which consequently results in an increased video rate delivered to the receivers. In light of these results, the remaining experiments in this section have all been carried out by enabling bandwidth probing at the conference controller.

6.7.2.2 Algorithm Comparison

In this section, we investigate the actual gains of the proposed framework using emulation. The results of the 24 different configurations summarized in Table 6.2 are reported in Figures 6.14 and 6.15. For the sake of brevity, we only report the results for the experiments with 28 receivers. Very similar results are obtained in the case with 10 receivers, which we therefore omit. Overall, the rate loss decreases



(a) Rate loss. (b) Played rate.

Figure 6.16: The performance reached by the proposed framework on the emulation testbed is worse compared to that obtained with the Java-based simulator, both in terms of average rate loss (6.16a) and played rate (6.16b).

by 20% when the ILP formulation is used. The gains are higher for a low number of encoders and when the optimization period is shorter. For instance, for the ILP formulation, the average rate loss improves by 23%, 19% and 17% when three, four and five encoders are available, respectively. These results clearly show the benefits of the proposed framework. In terms of played rate, the average gain when using the ILP formulation is about 11%. The results presented in this section thoroughly confirm the trends and analyses performed using the Java-based simulator and the gains brought by the proposed framework. In the next section, we highlight the main difference between emulation and simulation and the optimizations that can further improve the performance of the proposed framework.

6.7.2.3 Comparison Between Emulation and Simulation

In order to compare the results obtained with the Java-based simulator and the emulation testbed, we select a configuration with 28 receivers, five available encoders and eight seconds optimization period, for the ILP formulation. The same experiment configuration was repeated in both simulation and emulation. The correspondent results are presented in Figure 6.16. Compared to simulation, the rate loss increases by about 25 kbps for all the different approaches, while the average played rate decreases by almost 180 kbps in the emulation testbed. Nonetheless, the relative performance among the different approaches is very similar in both simulation and emulation. This difference is mainly due to the way the receivers' bandwidth is estimated in simulation and emulation. The WebRTC-like bandwidth estimation presented in Algorithm 1 is modeled after the real bandwidth estimation pattern of a classical WebRTC streaming, where the sender can probe the available bandwidth of the receiver by altering the encoding bitrate of the video itself. In emulation instead, probing traffic is artificially generated by the controller using

the procedure explained in Section 6.5.3. Although effective, as shown in Section 6.7.2.1, the receivers' bandwidth estimation performed at the conference controller in emulation evolves more slowly than in simulation and is on average lower. This aspect is directly reflected on the average rate loss and played rate of the receivers. This analysis points out the need for an efficient probing algorithm at the conference controller, that can further improve the end-to-end bandwidth estimation of the receivers.

6.8 Conclusions

In this chapter, a framework has been proposed for the efficient delivery of WebRTC streams in the context of remote video collaboration applications. Classical streaming techniques as HTTP adaptive streaming cannot guarantee the low latency and interactivity required in these scenarios. Consequently, the open-source browser-based WebRTC protocol has been used instead. In order to overcome the scalability issues of a classical WebRTC peer-to-peer architecture, only a few encoders are used at sender-side, where each encoder transmits to several WebRTC receivers at the same time. This choice allows to scale the proposed approach to a large number of receivers and reduce the encoding costs. A conference controller, implemented using the Jitsi-Videobridge software, dynamically forwards the most suitable stream to each receiver, to accommodate fast bandwidth variations and ensure a continuous playout. Besides this short-term adaptation, the controller periodically recomputes the set of encoding bitrates to better follow the long-term network conditions of the receivers. The optimal solution to the bitrate recomputation problem has been found using an ILP formulation when the number of receivers is small, and in an approximate but fast way using the K-means clustering algorithm. Both simulation and emulation results confirm the gains brought by the proposed approach, which was tested in a one-to-many remote collaboration scenario, typical in virtual classrooms, for example. In an emulated scenario with 28 receivers and one sender equipped with three encoders, the proposed framework can increase the delivered video rate up to 15%, compared to a static, fixed association of the encoding bitrates. Moreover, the dynamic recomputation is more efficient than a static approach, as less encoders are needed to obtain similar performance. For the same configuration mentioned above, similar results in terms of rate loss and played rate are obtained using a static association and four encoders at sender-side and the proposed dynamic recomputation with three encoders.

Acknowledgment

Jeroen van der Hooft is funded by grant of the Agency for Innovation by Science and Technology in Flanders (VLAIO). This research was performed partially within the imec PRO-FLOW project (150223).

References

[1] J. Jang-Jaccard, S. Nepal, B. Celler, and B. Yan. *WebRTC-Based Video Conferencing Service for Telehealth*. Computing, 98(1):169–193, Jan 2016.

[2] H. Oh, S. Ahn, J. Choi, and J. Yang. *WebRTC Based Remote Collaborative Online Learning Platform*. In Proceedings of the 1st Workshop on All-Web Real-Time Systems, AWeS '15, pages 9:1–9:5, New York, NY, USA, 2015. ACM.

[3] M. de Paiva Guimarães, D. Dias, J. Mota, B. Gnecco, V. Durelli, and L. Trevelin. *Immersive and Interactive Virtual Reality Applications Based on 3D Web Browsers*. Multimedia Tools and Applications, Dec 2016. Available from: https://doi.org/10.1007/s11042-016-4256-7, doi:10.1007/s11042-016-4256-7.

[4] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Journal of Network and Systems Management, pages 1–28, 2017.

[5] S. Loreto and S. P. Romano. *How Far Are We from WebRTC-1.0? An Update on Standards and a Look at What's Next*. IEEE Communications Magazine, 55(7):200–207, 2017.

[6] B. Grozev, L. Marinov, V. Singh, and E. Ivov. *Last N: Relevance-Based Selectivity for Forwarding Video in Multimedia Conferences*. In Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 19–24. ACM, 2015.

[7] Y. Xu, C. Yu, J. Li, and Y. Liu. *Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype*. IEEE/ACM Transactions on Networking, 22(3):826–839, June 2014.

[8] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano. *Performance Analysis of the Janus WebRTC Gateway*. In Proceedings of the 1st Workshop on All-Web Real-Time Systems, AWeS '15, pages 4:1–4:7, New York, NY, USA, 2015. ACM.

[9] L. López, M. París, S. Carot, B. García, M. Gallego, F. Gortázar, R. Benítez, J. A. Santos, D. Fernández, R. T. Vlad, and F. J. Gracia, .and López. *Kurento: The WebRTC Modular Media Server*. In Proceedings of the 2016 ACM on Multimedia Conference, MM '16, pages 1187–1191, New York, NY, USA, 2016. ACM.

[10] L. López-Fernández, B. García, M. Gallego, and F. Gortázar. *Designing and Evaluating the Usability of an API for Real-Time Multimedia Services in the Internet*. Multimedia Tools and Applications, 76(12):14247–14304, Jun 2017.

[11] L. Ma, D. Veer, W. Chen, G. Sternberg, Y. A. Reznik, and R. A. Neff. *User Adaptive Transcoding for Video Teleconferencing*. In 2015 IEEE International Conference on Image Processing (ICIP), pages 2209–2213, Sept 2015.

[12] J. C. Granda, P. Nuo, F. J. Surez, and D. F. Garca. *Overlay Network Based on WebRTC for Interactive Multimedia Communications*. In 2015 International Conference on Computer, Information and Telecommunication Systems (CITS), pages 1–5, July 2015.

[13] M. A. Hossain and J. I. Khan. *Distributed Dynamic MCU for Video Conferencing in Peer-to-Peer Network*. In 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC), pages 1–8, Dec 2016.

[14] P. Rodrguez, lvaro Alonso, J. Salvacha, and J. Cervio. *Materialising a New Architecture for a Distributed MCU in the Cloud*. Computer Standards and Interfaces, 44(Supplement C):234 – 242, 2016.

[15] J. Trnkoczy, U. Pacinski, S. Gec, and V. Stankovski. *SWITCH-ing from Multi-Tenant to Event-Driven Videoconferencing Services*. In 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pages 219–226, Sept 2017.

[16] P. Rodríguez, A. Alonso, J. Salvachúa, and J. Cervino. *dOTM: A Mechanism for Distributing Centralized Multi-party Video Conferencing in the Cloud*. In 2014 International Conference on Future Internet of Things and Cloud, pages 61–67, Aug 2014.

[17] M. Wenzel and C. Meinel. *Full-Body WebRTC Video Conferencing in a Web-Based Real-Time Collaboration System*. In 2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 334–339, May 2016.

[18] V. Xhagjika, . D. Escoda, L. Navarro, and V. Vlassov. *Load and Video Performance Patterns of a Cloud Based WebRTC Architecture*. In 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pages 739–744, May 2017.

[19] E.-z. Yang, L.-k. Zhang, Z. Yao, and J. Yang. *A Video Conferencing System Based on SDN-Enabled SVC Multicast*. Frontiers of Information Technology & Electronic Engineering, 17(7):672–681, Jul 2016.

[20] B. Grozev, G. Politis, E. Ivov, T. Noel, and V. Singh. *Experimental Evaluation of Simulcast for WebRTC*. IEEE Communications Standards Magazine, 1(2):52–59, 2017.

[21] J. D. Praeter, H. Swimberghe, G. Renard, G. V. Wallendael, and P. Lambert. *Dynamic encoder profile optimisation for real-time video streaming applications*. Electronics Letters, 52(13):1116–1118, 2016.

[22] S. Petrangeli, D. Pauwels, J. van der Hooft, J. Slowack, T. Wauters, and F. De Turck. *Dynamic Video Bitrate Adaptation for WebRTC-Based Remote Teaching Applications*. In 2018 IEEE Network Operations and Management Symposium (NOMS), May 2018.

[23] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. *Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC)*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 13:1–13:12, New York, NY, USA, 2016. ACM.

[24] H. Alvestrand. *RTCP message for Receiver Estimated Maximum Bitrate*. Internet-Draft draft-alvestrand-rmcat-remb-03 (work in progress), 2013.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. *The WEKA Data Mining Software: An Update*. SIGKDD Explor. Newsl., 11(1):10–18, November 2009.

[26] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-based Bandwidth-Lookup Service for Bitrate Planning*. ACM Transactions on Multimedia Computing, Communications and Applications, 8(3):24:1–24:19, August 2012.

[27] D. Schroeder, A. E. Essaili, E. Steinbach, D. Staehle, and M. Shehada. *Low-Complexity No-Reference PSNR Estimation for H.264/AVC Encoded Video*. In 2013 20th International Packet Video Workshop, pages 1–6, Dec 2013.

7 Conclusions

"Quod scripsi, scripsi."

–John 19:22

In this dissertation, several optimizations have been proposed for the efficient delivery of adaptive video streaming services. These solutions make extensive use of network components, developed and deployed to support the video clients and improve their final Quality of Experience (QoE). In this section, we review the challenges addressed in this dissertation and outline several research directions currently arising in the field of multimedia delivery, which we believe will be relevant in future years.

7.1 Review of the Addressed Challenges

Each of the chapters of this dissertation have addressed one of the challenges discussed in Chapter 1, as summarized in the following.

Challenge #1: provide a video streaming service that is fair to the end user from the QoE point of view. To this aim, we have proposed in Chapter 3 an in-network system of *coordination proxies*, coupled with a new client-based rate adaptation heuristic. Classical HTTP Adaptive Streaming (HAS) systems can suffer from fairness problems when multiple clients stream a video at the same time using the same bottleneck links. To relieve this issue, the in-network system computes an estimate of the fair bandwidth share among the competing clients, and

communicates this measurement to the video clients, which do not need to communicate among each other. This fairness value is used by the clients to request the video quality that can both optimize their own QoE while being fair to the other users. Several experiments have been carried out to test the proposed in-network system. Particularly, the solution is able to increase the average QoE by almost 20% while improving fairness, computed as the QoE standard deviation, with almost 80% compared to several benchmarking heuristics. The proposed approach has also been shown to be robust towards malicious clients that deliberately neglect to collaborate and towards malfunctioning of the in-network system. These results have been obtained without negatively impacting the performance of other adaptation heuristics, in an heterogeneous scenario where different clients are equipped with different heuristics.

Challenge #2: avoid annoying interruptions, also called freezes, during the video playback. Chapter 4 therefore has presented a network controller, based on the Software-Defined Networking (SDN) principle and OpenFlow, that can temporarily prioritize the delivery of particular video segments to avoid the occurrence of a freeze. The network controller intercepts the HTTP GET requests sent by the clients when requesting a new segment and, based on these requests, estimates the clients' buffer status. Using this information, the requested quality and the network bandwidth, a prioritization algorithm, located at the controller, decides whether the requested segment should be prioritized or not. The proposed framework does not require an explicit communication between controller and clients and can optimize the behavior of any adaptation heuristic. Results obtained on an emulation setup implemented using state-of-the-art software have confirmed the benefits of prioritization. A dimensioning study has been initially performed, which allowed to guarantee that the proposed prioritization system does not negatively influence the performance of cross-traffic applications, modeled as web-browsing and progressive download clients. Next, a comparison has been performed with several HAS heuristics, which showed that the prioritization controller can almost completely eliminate video freezes in a wide range of scenarios, with varying cross-traffic loads, network bandwidth and client configurations. Moreover, a scalability study has confirmed that the proposed system can control and optimize the performance of several thousand clients at the same time.

Challenge #3: increase the adaptability of the network elements to support unseen video streaming scenarios, reducing human hand-tuning. For this reason, a machine learning-based prioritization controller has been developed in Chapter 5, which autonomously learns when a video segment should be prioritized. Despite the good performance of the priorization algorithm presented in Chapter 4, the proposed prioritization system requires a certain degree of tuning and relies on specific estimated measurements obtained from the video clients (e.g., the buffer level). Such a hard-coded approach might be sub-optimal in certain streaming sce-

narios. Therefore, a machine learning algorithm based on Random Undersampling Boosting (RUSBoost) has been used in Chapter 5 to perform the prioritization task. The algorithm is trained off-line to correlate information such as the network bandwidth and the inter-arrival time of consecutive HTTP GET requests to the occurrence of video freezes. The resulting prioritization policy can successfully foresee the occurrence of freezes not only when the streaming conditions are similar to the training settings, but also in case of unseen videos, client heuristics and network topologies. Compared to a baseline heuristic not optimized by prioritization, the machine learning approach has been able to reduce video freeze by 65% and freeze time by 45%. Moreover, it has been shown that a distributed network of independent prioritization controllers is still able to provide good performance even in a multi-bottleneck network topology.

Challenge #4: reduce costs and increase scalability and performance for interactive streaming applications. Therefore, Chapter 6 has presented a scalable Web Real-Time Communication (WebRTC) framework for remote video collaboration applications, as remote teaching or telehealth. HAS techniques cannot be employed in this case, as remote collaboration applications require very low-latency and interactivity. Therefore, the WebRTC standard has been used in Chapter 6. Since WebRTC is peer-to-peer by design, scalability problems can arise when a large number of peers participate at the same remote session. In classic WebRTC, the sender would need to encode a separate, independent stream for each receiver. To relieve this scalability issue, a framework has been proposed to support the delivery of real-time WebRTC streams. In the proposed framework, the sender only needs to encode a limited amount of streams at different bitrates. A conference controller receives these streams and forwards them to the receivers based on their bandwidth conditions. Moreover, the controller periodically recomputes the encoding bitrates of the sender, using an Integer Linear Programming (ILP) formulation, so that the bitrates are close and representative of the bandwidth conditions of the receivers. The proposed framework has been tested in a one-to-many remote collaboration scenario, where most of the interaction among the participants is dominated by a single peer. As an example, this interaction is typical in a virtual classroom, where most of the communication occurs from the lecturer to the students. Simulation and emulation results have confirmed the gains of the proposed approach. Particularly, in a scenario with 28 receivers and three encoders at sender-side, the proposed framework has been able to increase the rate obtained by the receivers by almost 15%, compared to a solution where the encoding bitrates are fixed and do not change over time. Moreover, the proposed dynamic recomputation has proven to be more cost-efficient than a static association, meaning that similar performance can be obtained using less encoders at sender-side.

In addition to the above mentioned challenges, Chapter 2 presented an elaborated survey of the research efforts carried out in the adaptive video streaming

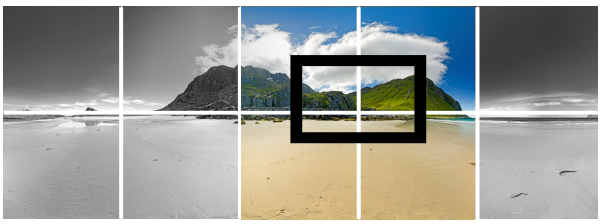


Figure 7.1: In tiled VR streaming, the 360° video is divided into spatial regions. Only tiles belonging to the viewport are streamed at the highest quality, to save bandwidth.

domain, with a particular focus on network- and service-assisted solutions, and optimizations taking place at the application or transport layer. Moreover, a set of guidelines and recommendations have been given to identify which solution is more tailored to improve specific QoE factors, as freezes or video quality, and its deployment complexity.

7.2 Future Directions and Challenges

As reported and analyzed throughout this PhD thesis, several consistent improvements have been developed to optimize the QoE of adaptive streaming services. Nevertheless, several new challenges are emerging, which will need to be addressed by the multimedia management community in the near future. In this section, the main challenges and the newly arising opportunities associated with them are identified.

7.2.1 Immersive Video Streaming

Virtual Reality (VR) devices are quickly becoming accessible to a large public. It is therefore expected that the demand for 360° immersive videos will grow consistently in the next years. In VR streaming, the user is immersed in a virtual environment and can dynamically and freely decide the preferred viewing position, called viewport. Unfortunately, VR streaming is often affected by low quality nowadays, due to the high bandwidth requirements of 360° videos. Viewport-dependent solutions have often been proposed for VR streaming, as they are able to reduce the bandwidth required to stream the VR video [1, 2]. In viewport-dependent streaming, only the portion of the video actually watched by the user is streamed at the highest quality. The rest of the video, which is outside the viewport and is therefore less important, can be streamed at a lower quality or not streamed at all. Viewport-dependent streaming can be obtained using online transcoding operations, as foveat-based encoding [1], or by spatially tiling the video [2], as shown in Figure 7.1. This last possibility is extremely interesting in the HAS domain, as

MPEG-DASH has recently standardized a new specification, called spatial representation description, to support tile-based streaming [3]. In the context of tiled VR streaming, three important questions should be answered. First, *how to prioritize the delivery of viewport tiles as opposed to tiles outside the viewport?* Clearly, tiles in view should be delivered faster than the others. The new HTTP/2 protocol, with its new features as server push, stream prioritization and multiplexing, can represent a possible solution [4]. Second, *how to predict where a user is going to watch in the near future?* When moving, the user can reach regions of the video at lower qualities. It is therefore important to predict these changes in order to provide a seamless transition when the user moves. As an example, Fan et al. propose a recurrent neural network to estimate the fixation point for 360° videos [5]. Third, *what is the effect on QoE of watching regions at different qualities?* Despite the benefits of prediction algorithms, the user can still be presented with a viewport at different quality levels. The impact of this behavior on QoE should be investigated to provide an important input for future tile-based rate adaptation heuristics.

7.2.2 HTTP Adaptive Streaming over QUIC

As discussed in Chapter 2, several advantages can be obtained when the transport layer is modified to better support streaming traffic. Unfortunately, TCP modifications are difficult to deploy, as TCP resides in the system kernel of a device. To relief this issue, Google has recently proposed a new transport layer protocol, called the Quick UDP Internet Connection (QUIC) protocol [6]. Being based on UDP, QUIC can be implemented in the user space rather than the kernel, and can therefore be deployed and updated more easily than TCP. To guarantee reliability, QUIC has to implement a congestion control algorithm, similarly as for TCP. Any kind of congestion control algorithms can be implemented and, therefore, also those already developed for TCP, as CUBIC [7] or the new BBR algorithm proposed by Google [8]. QUIC provides several interesting improvements compared to TCP. First, 0-RTT connection establishment when client and server have already communicated in the past, which helps reducing latency. Second, true multiplexing of HTTP/2 streams at the transport level, as opposed to standard TCP that still introduces head-of-line blocking when the packets of a certain stream are lost. Third, the possibility to easily deploy new congestion control algorithms. The Google QUIC team reports a rebuffering rate reduction of YouTube playbacks by 18.0% for desktop users and 15.3% for mobile users [9], when using QUIC compared to standard TCP. Moreover, the number of videos played at their optimal rate increases by 2.9% for desktop and by 4.6% for mobile users, respectively. These results have been obtained by deploying the QUIC protocol globally for Google services. As pointed out by this work, QUIC introduces major innovations that can be useful in HAS, and is especially useful whenever network congestion, loss, and

RTT are high. Nevertheless, a formal analysis of adaptive streaming over QUIC has only been marginally investigated [10], and is still missing at the moment. In light of the above, *what is the impact of QUIC on adaptive streaming and in which network conditions does it outperform standard TCP?* Moreover, *is it possible to develop a congestion control algorithm tailored on the adaptive streaming needs?* TCP modifications are somewhat limited by the actual applicability of the solution and the a priori limitations of TCP. As listed above, QUIC opens up a new range of possibilities to develop real HAS-aware congestion control algorithms.

7.2.3 Traffic Encryption

Nowadays, privacy and security have become two of the main requirements for Internet users. Therefore, it is expected that an increasingly larger portion of Internet traffic will be encrypted in the next few years. As an example, QUIC traffic is encrypted by design while HTTP/2 is only supported with encryption enabled by web browsers. This trend concerns video streaming applications as well. Despite being beneficial from a user perspective, traffic encryption can pose a serious challenge on network operations. Most of the network-assisted solutions developed to optimize adaptive video streams rely on the possibility of intercepting and analyzing video traffic to carry out their optimizations. Encryption will make these solutions more and more difficult to apply in future years, unless the streaming provider and the network provider overlap or agree to collaborate. In order to allow network-based solutions to fully optimize HAS streams, QoE factors would need to be estimated. Consequently, *is it possible to infer QoE factors inside the network for encrypted HAS traffic?* Machine learning algorithms will play a very relevant role in this case. For instance, Dimopoulos et al. have shown that it is possible to classify QoE events for YouTube encrypted traffic, in terms of stalls, quality switches and average quality [11]. A similar conclusion is also drawn by Orsolic et al. [12]. A strong research focus is expected in this area, to create prediction algorithms that can classify encrypted traffic online and with high accuracy, two requirements for the applicability of network-assisted solutions. Moreover, the impact of encryption on user experience has not been investigated so far. In other words, *what is the contribution of encryption in video streaming on the users' QoE?* For example, a user might be willing to trade a certain level of encryption in exchange of a superior QoE, which can be provided by using network solutions. In the future, we can expect personalized QoE models including privacy and security together with classical video streaming metrics.

7.2.4 Personalized QoE-Centric Control

As reported in Chapter 2, a large body of research has investigated how to improve the delivery of adaptive video streams, taking QoE parameters explicitly into ac-

count. Despite that, these works still suffer from three inefficiencies. First, the employed QoE models are developed to capture the behavior of the "average" user, and are therefore not personalized. Second, standard models do not consider the context in which the streaming session takes place. Third, only the QoE model of the users is inserted into the control loop, but not the user itself. In other words, the actual online user experience is not captured and is not used in the online optimization of the streaming service. Three challenges can be identified in the domain of personalized QoE-centric control. First, *how can a QoE model for HAS, which is representative of the specific, rather than aggregate, user behavior be created?* Estimating QoE does not only involve application-level parameters, such as switches or freezes, but also sensorial inputs and context [13, 14]. Current smartphones and tablets are already equipped with several sensors (e.g., GPS, light sensors etc.), and wearable devices will allow to obtain even more fine-grained information on the user status (e.g., heart level, eye-tracking etc.). Using both network-, application- and user-level parameters will allow to create real personalized QoE models. Second, *where should the QoE model computation be carried out?* Machine learning algorithms represent a good candidate for the correlation of the aforementioned different parameters. As the input space increases though, it could become impossible to learn such a complex model directly on the user's device. Offloading such computational intensive task to the cloud can represent a viable solution. In this scenario, the real-time delay restrictions of the QoE modeling task should be carefully taken into account. Third, *how can the online feedback of the user be included into the control of the adaptive streaming service?* This feedback can be both explicit, if the user can directly rate the viewing experience, and implicit, when the sensorial information is used to estimate the personalized user's QoE. Particularly, a complex model that encompasses not only application-level but also user-level parameters could be used to estimate on-the-fly and online the real QoE of the specific user. This aspect opens up the possibility to control the service directly at the user level, in order to enforce specific QoE levels that are representative of the real way the user perceives the video streaming session.

7.2.5 Video Delivery for Low-Latency and High-Mobility Applications

Low-latency streaming applications, as immersive streaming, gaming and real-time communication, will become dominant in future years. Moreover, the current trend for telecommunication networks is to evolve towards large scale deployments that can encompass billions of devices, the so-called Internet of Things (IoT). Many of these devices will have streaming capabilities as well. For instance, as vehicles are becoming smart and connected, in-car entertainment services and applications will gain momentum. Swarms of drones are already being deployed to

monitor and support the coverage of live events or emergency operations. All these scenarios can be characterized by a high degree of mobility, low-latency or reliability requirements, or a combination thereof. Even though mobility and latency have already been investigated in the adaptive streaming domain, a shift occurs in the delivery architecture of these new applications. As an example, autonomous cars can both stream from a stable network infrastructure or from each other [15]. In light of the above, *how can continuous, reliable and low-latency video streams be provided in an IoT scenario?* Adaptive video streaming will have a central role to enable these dynamic services, especially when combined with new network paradigms as 5G and softwarized networks [16]. 5G will be able to support both traditional forms of communication and more unstructured ones, as machine-to-machine communication. This aspect entails that the network has to be flexible and able to dynamically reconfigure itself, depending on the application and its requirements. Context-awareness, both applicable in the video client and in the network, will also be crucial in this domain. In such a dynamic ecosystem, clients' conditions can drastically change or be extremely unreliable. Being aware of these characteristics will be essential to provide the best service to the final users.

7.2.6 Open Software and Dataset Availability

One of the efforts that was carried out during this thesis was the development of emulation software to compare the performance of the proposed solutions with state-of-the-art algorithms. One of the main procedural challenges that complicate this task is the lack of a common open-source platform or framework that would allow to compare different strategies among each other. Some attempts in this direction have already been carried out, for example by De Cicco et al. and Schwarzmann et al. [17, 18]. The former allows to easily compare different rate adaptation heuristics, while the latter allows to compare network-assisted strategies. Stohr et al. implements an emulation environment for the systematical comparison and analysis of different DASH players and rate adaptation heuristics [19]. Moreover, open access to experimentation facilities and testbeds[1] can stimulate research efforts in this direction. Public datasets, in terms of HAS videos [20] or bandwidth conditions [21, 22], are also very important when it comes to the replicability of results. This trend should be encouraged in future years, in order provide a common ground for the development and comparison of new and existing solutions in the adaptive streaming domain.

[1] https://www.fed4fire.eu/

References

[1] J. Ryoo, K. Yun, D. Samaras, S. R. Das, and G. Zelinsky. *Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 6:1–6:11, New York, NY, USA, 2016. ACM.

[2] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. *Optimizing 360 Video Delivery over Cellular Networks*. In Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges, ATC '16, pages 1–6, New York, NY, USA, 2016. ACM.

[3] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim. *MPEG DASH SRD: Spatial Relationship Description*. In Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, pages 5:1–5:8, New York, NY, USA, 2016. ACM.

[4] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck. *An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos*. In Proceedings of the 2017 ACM on Multimedia Conference, MM '17, pages 306–314, New York, NY, USA, 2017. ACM.

[5] C. Fan, J. Lee, W. Lo, C. Huang, K. Chen, and C. Hsu. *Fixation Prediction for 360° Video Streaming in Head-Mounted Virtual Reality*. In Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV'17, pages 67–72, New York, NY, USA, 2017. ACM.

[6] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kuehlewind. *Innovating Transport with QUIC: Design Approaches and Research Challenges*. IEEE Internet Computing, 21(2):72–76, Mar 2017.

[7] S. Ha, I. Rhee, and L. Xu. *CUBIC: A New TCP-friendly High-speed TCP Variant*. SIGOPS Oper. Syst. Rev., 42(5):64–74, July 2008.

[8] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. *BBR: Congestion-based Congestion Control*. Commun. ACM, 60(2):58–66, January 2017.

[9] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. Chang, and Z. Shi. *The QUIC Transport Protocol: Design and Internet-Scale Deployment*. In Proceedings of the Conference of the ACM Special Interest Group

on Data Communication, SIGCOMM '17, pages 183–196, New York, NY, USA, 2017. ACM.

[10] C. Timmerer and A. Bertoni. *Advanced Transport Options for the Dynamic Adaptive Streaming over HTTP*. CoRR, abs/1606.00264, 2016. Available from: http://arxiv.org/abs/1606.00264.

[11] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki. *Measuring Video QoE from Encrypted Traffic*. In Proceedings of the 2016 Internet Measurement Conference, IMC '16, pages 513–526, New York, NY, USA, 2016. ACM.

[12] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov. *A machine learning approach to classifying YouTube QoE based on encrypted network traffic*. Multimedia Tools and Applications, May 2017.

[13] B. Rainer and C. Timmerer. *A Generic Utility Model Representing the Quality of Sensory Experience*. ACM Trans. Multimedia Comput. Commun. Appl., 11(1s):14:1–14:17, October 2014.

[14] P. Reichl, S. Egger, S. Moeller, K. Kilkki, M. Fiedler, T. Hossfeld, C. Tsiaras, and A. Asrese. *Towards a Comprehensive Framework for QoE and User Behavior Modelling*. In 2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX), pages 1–6, May 2015.

[15] H. He, H. Shan, A. Huang, and L. Sun. *Resource Allocation for Video Streaming in Heterogeneous Cognitive Vehicular Networks*. IEEE Transactions on Vehicular Technology, 65(10):7917–7930, Oct 2016.

[16] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. *Network Slicing in 5G: Survey and Challenges*. IEEE Communications Magazine, 55(5):94–100, May 2017.

[17] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. *TAPAS: A Tool for rApid Prototyping of Adaptive Streaming Algorithms*. In Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming, VideoNext '14, pages 1–6, New York, NY, USA, 2014. ACM.

[18] S. Schwarzmann, T. Zinner, and O. Dobrijevic. *Towards a Framework for Comparing Application-Network Interaction Mechanisms*. In 2016 28th International Teletraffic Congress (ITC 28), volume 03, pages 13–18, Sept 2016.

[19] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg. *Where Are the Sweet Spots?: A Systematic Approach to Reproducible DASH*

Player Comparisons. In Proceedings of the 2017 ACM on Multimedia Conference, MM '17, pages 1113–1121, New York, NY, USA, 2017. ACM.

[20] S. Lederer, C. Müller, and C. Timmerer. *Dynamic Adaptive Streaming over HTTP Dataset*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 89–94, New York, NY, USA, 2012. ACM.

[21] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning*. ACM Trans. Multimedia Comput. Commun. Appl., 8(3):24:1–24:19, August 2012.

[22] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Alface, T. Bostoen, and F. De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks*. IEEE Communications Letters, 20(11):2177–2180, Nov 2016.